WIRELESS SENSOR NETWORK IMPLEMENTATIONS ON A TESTBED PLATFORM

BY

KYLE A. HARRIS

THESIS

Submitted in partial fulfillment of the requirements
for the degree of Master of Science in Electrical and Computer Engineering
in the Graduate College of the
University of Illinois at Urbana-Champaign, 2013

Urbana, Illinois

Adviser:

Professor Venugopal V. Veeravalli

ABSTRACT

Wireless sensor network implementations are developed for three different problems: data efficient quickest change detection, energy efficient sleep control for application to the tracking problem, and diffusing point source localization and tracking. For each of these three problems work in the field is briefly examined before simulations and implementations are presented.

Particular emphasis is placed on the tracking problem where a small extension is made to previous work in the field. A sub-optimal algorithm is developed that is computationally feasible for implementation purposes. This solution is compared to the more optimal solution already derived, and is then implemented on a testbed network of wireless sensors. Results from the testbed are compared, and shown to be superior in performance, to duty cycling, which is the current standard for power efficiency in a sensor network.

The implementations of the data efficient quickest change detection and the energy efficient sleep control for tracking algorithms both work well, and encourage immediate development for applied use. The implementation for the diffusing point source localization and tracking algorithm is by comparison less ready for deployment, and requires further development before applied use can be seriously considered.

ACKNOWLEDGMENTS

I want to thank my advisor, Prof. Venugopal Veeravalli, for all of the opportunities and guidance he made available to me. I thank my research group members for the encouragement, advice, and constructive criticism they shared. I want to thank my parents as well, for the constant love and support always provided. Finally thanks to Maple, for her ever present overabundance of enthusiasm and joy, and for brightening my life every day.

TABLE OF CONTENTS

CHAPTER 1

INTRODUCTION

In recent years technological development has allowed for the construction of very small and cheap wireless sensor nodes in large quantities. Although limited in capabilities on their own, when networked together these nodes become the elements of a wireless sensor network (WSN), which has a far greater value than the sum of its parts. The abilities of WSNs have only been briefly explored, but already include applications in wildlife habitat monitoring [1], forest fire detection [2], and structural health monitoring [3], [4]. As development of the hardware continues to advance, and our world becomes even more interconnnected with itself, the desirability and feasibility of larger and more complex implementations of these WSNs will continue to rise. This development makes WSNs an area of considerable interest to researchers, where much work remains to be done.

The focus of this work is to take several algorithms designed for use with sensor motes and networks and implement them on a testbed system. This implementation phase is a critical one to ensure the algorithm is in a good final state where it can be easily adapted, and used for the benefit of society. Many times issues can crop up in implementation, causing an algorithm to require refinement or adjustments to make it practical for larger applications. Implementation can be a tedious step in research, and is many times overlooked in favor of pursuing a new theoretical topic. This is unfortunate, as many times much valuable insight into the problem can often be gained even from the simplest of implementations.

Long node battery life is of great importance in the application of WSNs. Replacing the batteries of the entire network can be a tedious, difficult, and expensive endeavor depending on various influencing factors such as: the number of nodes in the network, the environment the network is located in, and the size of the area covered by the network. This implies that it is beneficial to replace the batteries of the nodes as infrequently as possible.

The use of intelligent sleep control schemes can greatly increase the energy efficiency of a WSN, increasing the battery life of the constituent nodes given a particular battery capacity.

Energy efficiency is the goal of the first two algorithms implemented in this thesis. They focus on improving the battery life of sensor motes in a network using the sensor observations for more intelligent control. The third section is the use of a sensor network to identify the position of a fixed diffusing point source emitter, and to track its output intensity over time. All three are algorithms that were first developed by different graduate students working for Professor Veeravalli at the University of Illinois at Urbana-Champaign. None of the algorithms were implemented, providing the motivation for this thesis.

Note that throughout this thesis, only WSNs in a centralized setting are being considered. This means that the data gathered by the WSN is all going to a common central control node, which makes decisions based on the information provided to it.

1.1 Overview

The most popular current method of sleep control in use by WSNs is simple duty cycling of the constituent nodes. In this method each node has a predetermined amount of time it will sleep after taking a measurement and reporting this back to the central controller. Duty cycling is very effective as an initial effort in reducing WSN power consumption, but does not use any of the information available to the WSN, leaving the door open for a more complex, effective control scheme. This opening is targeted in the first two algorithms implemented.

The first section of the thesis looks at an energy efficient extension [5], [6], [7] to the classic quickest change detection problem [8], [9]. The problem setup and algorithm description as given in [6] are briefly described, and then simulations of the algorithm are performed. These provide a basis for rough comparison with the implementation of the algorithm on the testbed, which concludes the first chapter of the thesis.

Sleep control is studied in some detail in a series of papers [10], [11], [12], [13], where the authors investigate the problem in context to an ob-

ject tracking WSN. In [11] the authors develop a simple framework for the object tracking problem. The space over which the object is being tracked is partitioned into a number of cells, and a sensor is located in each cell. The tradeoff in this setup is how many sensors need to be active each time step to accurately track the object as it moves through the network. The fewer the number of sensors active, the lower the WSNs power consumption, but this also generally implies a lower object tracking accuracy. The authors frame the problem in terms of a partially observable Markov decision process (POMDP) and then go on to reframe this problem in terms of a sufficient statistic to ensure unbounded memory is not necessary at each of the sensors to solve the problem. Using dynamic programming and some sub-optimal simplifications the authors arrive at a few different algorithms with varying degrees of effectiveness at solving the sleep control problem in an energy efficient way.

The authors develop a more generalized model for the observations taken by the sensors, and allow for a more complex discrete state space representation of the objects position in [12]. Now they assume that the object can take positions independent of the sensor locations, and that each sensor now takes a discrete or possibly continuous measurement based on an observation model. This ends up allowing for a much more general network setup than is posed in [11], and better reflects real world implementations. The process of finding a solution is handled in an almost identical series of steps in [11] and [12], beginning again with the problem setup as a POMDP, finding a sufficient statistic, and then using dynamic programming and a sub-optimal simplification to solve the problem.

One major contribution of this thesis is to take the problem as posed in [12], and develop a very simple algorithm to arrive at a good approximate solution. This algorithm ideally performs as well as the solution in [12], but does not require nearly as much computational complexity. This algorithm is developed using insights gained through the study of simulations of WSNs using the method in [12], but specifically avoiding the use of dynamic programming to arrive at an approximate solution.

Simulations of this simplified algorithm show that it performs almost as well as the solutions in [12], in multiple different sensor array setups. Results for a 1-dimensional sensor array, a 2-dimensional grid array, and a 2-dimensional randomized sensor location array are all compared for the sleep

control schemes of duty cycling, the method used in [12], and the simplified algorithm. Finally, A WSN is developed and implementations of the duty cycling scheme and the new simple algorithm are compared. The tests are done for 1-dimensional sensor array, and a 2-dimensional grid array. These results are not as nice as the results from the simulations, but this is to be expected considering all the non-idealities present in an implemented testbed.

The last chapter of this thesis looks at the problem of localizing and intensity tracking of a diffusing point source using sensor networks, as given in [14]. This is an interesting application of an algorithm using Kalman filtering and the Recursive Prediction Equations [15] as its basis to a problem using sensor networks. An interesting application that comes to mind is the localization of a polluting point source, and the tracking of its output over time. This could be very useful to cities with poor air quality that are trying to identify and monitor the primary polluters causing public health issues.

This chapter is broken up into a brief overview of the problem setup and algorithm description as given in [14], followed by simulations of the algorithm. The simulations are designed to match what the original author did as closely as possible to ensure the algorithm is functioning correctly. This algorithm is then ported over to an implemented sensor network testbed tracking using temperature as a diffusing medium.

Finally, all the contributions of this work are summarized, and areas for future work are identified in the conclusion.

1.2 Notation

Here we define some of the notation used through the duration of this thesis.

- Scalars and scalar-valued random variables are written in lower case (e.g. c, r).
- Matrices are written in upper case (e.g. P).
- Sets are denoted in calligraphic font (e.g. $\mathcal{B}$).
- $\mathbb{1}_{\{\bullet\}}$ denotes the indicator function.
- The vector $\boldsymbol{e}_i$ has a one in the i^th position, and zeros everywhere else.

CHAPTER 2

DATA-EFFICIENT QUICKEST CHANGE DETECTION

The classic quickest change detection problem is laid out in [8], [9]. This problem is expanded upon in [5], [6], [7] by introducing the notion of energy efficiency. An algorithm called Data-Efficient Cumulative Summation (DE-CUSUM) is developed that is shown to be asymptotically optimal under both the Bayesian and the non-Bayesian Minimax formulations [5], [6].

This chapter is composed of a quick problem setup and algorithm overview for DE-CUSUM as given in [6], followed by results from simulation and implementation of the algorithm on a physical sensor testbed using a photodiode to detect a change in light level.

2.1 Problem Setup

In the classic quickest change detection problem it is assumed that there is some system which generates a random process of independent identically distributed (i.i.d.) measurements X_k for time steps $k = 1, 2, ...$ which follow some pre-change distribution f_0. Then at some point in time, Γ, a change in the system occurs and the measurements now follow an i.i.d. post-change distribution f_1. The goal of the basic quickest change detection algorithm is to determine as quickly as possible after Γ that a change has occurred. Note the subtle difference between this goal and the attempt to actually determine Γ itself. The Cumulative Summation (CUSUM) algorithm [8] turns out to be the optimal test for this classic problem setup.

The energy efficient extension of the classic problem given in [6] takes into consideration energy efficiency by attempting to reduce the number of measurements taken while the system is in the pre-change state. The pre-change duty cycle (PDC) is the frequency at which the sensors take readings while the system is in the pre-change state. The conditional average detection

delay (CADD) is the average time it takes for the test to determine that the change has occurred starting from immediately after the change has occurred. The false alarm rate (FAR) for a test is the probability the test decides the change has occurred prior to it actually occurring. The goal of this modified problem is to minimize CADD while keeping the FAR for the test below a certain threshold, and the PDC below a different threshold. The problem is more formally defined in the algorithm description section that follows. In [5] and [6] the authors show that the DE-CUSUM algorithm is asymptotically optimal for the problem in both the Bayesian (where a prior for the change-point time Γ is assumed to be a gamma distribution and is known), and the Minimax (in which no prior for the change-point time is known) settings.

2.2 Algorithm Description

The DE-CUSUM achieves its energy efficiency by using controlled sensing to put the sensor in a low power sleep state when the algorithm believes with high likelihood the system is in the pre-change state. Intuition suggests that the higher the likelihood our system is in the pre-change state, the longer the sensor can afford to sleep before taking another measurement. At every time step k, let $S_k = 0$ indicate a measurement X_k will not be taken, and let $S_k = 1$ indicate X_k is to be taken. S_k is a binary control input based on the total information I available up to time $k-1$,

$$S_k = \nu_{k-1}(I_{k-1}), k = 1, 2, ...,$$

with ν denoting the control law and information vector

$$I_k = \left[S_1, ..., S_k, X_1^{(S_1)}, ..., X_k^{(S_k)}\right],$$

with $X_i^{(S_i)}$ representing X_i if $S_i = 1$ and X_i being absent from the vector I_k if $S_i = 0$. $\gamma = \{\tau, \nu_0, ..., \nu_{\tau-1}\}$ represents a policy for data-efficient quickest change detection where τ is the stopping time for the information sequence $\{I_k\}$.

The PDC, CADD and FAR are mathematically defined as follows:

$$\text{PDC} = \limsup_n \frac{1}{n} \mathbb{E}_n \left[\sum_{k=1}^{n-1} S_k \middle| \tau \geq n \right]$$

$$\text{CADD}(\gamma) = \sup_n \mathbb{E}_n[\tau - n | \tau \geq n]$$

$$\text{FAR} = \frac{1}{\mathbb{E}_\infty[\tau]},$$

where $\mathbb{E}_n[\bullet]$ indicates the conditional expectation of $\bullet$ when the change occurs at time n.

Adjusting the control law ν essentially adjusts the PDC. The trade-off is that lowering the PDC will increase the CADD, and the quickest change detection problem is to solve the following optimization problem:

$$\min_\gamma \text{CADD}(\gamma),$$

$$\text{subject to FAR}(\gamma) \leq \zeta, \text{and PDC}(\gamma) \leq \eta,$$

with the ζ and η constraints being desired performance parameters.

W_k is a sufficient statistic for the measurements $\{X_1, ..., X_k\}$ that is to be used to determine whether a change has happened or not. Given the pre- and post-change distributions on the measurements X_k (f_0 and f_1 respectively), the DE-CUSUM algorithm as described in [6] works as follows:

First start with the statistic $W_0 = 0$. Fix the parameters $\mu > 0$, $d > 0$ and $h \geq 0$. For time $k \geq 0$ use the following control:

$$S_{k+1} = \begin{cases} 0 & \text{if } W_k < 0 \\ 1 & \text{if } W_k \geq 0 \end{cases}$$

$$\tau_w(d) = \inf \{k \geq 1 : W_k > d\}.$$

The statistic W_k is updated using the following recursions:

$$W_{k+1} = \begin{cases} \min \{W_k + \mu, 0\} & \text{if } S_{k+1} = 0 \\ (W_k + \log L(X_{k+1}))^{h+} & \text{if } S_{k+1} = 1 \end{cases},$$

where $L(X_{k+1}) = f_1(X_{k+1})/f_0(X_{k+1})$, and

$$(x)^{h+} = \begin{cases} \max\{x, 0\} & \text{if } x > -h \\ x & \text{otherwise} \end{cases}.$$

From this description it becomes obvious how larger μ parameter values will increase the PDC while decreasing CADD, and how larger d values will decrease the FAR while increasing CADD. The term h is a parameter that adjusts the minimum undershoot necessary for the statistic W_k before the sensor sleeping takes effect. Setting $h = \infty$ causes the DE-CUSUM to be equivalent to the CUSUM algorithm, and setting $h = 0$ means DE-CUSUM will cause the sensor to sleep any time the statistic W_k drops below zero. Adjusting h allows for arbitrary PDC rates.

2.3 Simulation Results

The simulations of the DE-CUSUM algorithm are all shown relative to the evolution of the statistic for the basic CUSUM algorithm for an optimal basis of comparison. The simulations are all done using Matlab R2011b.

2.3.1 Simulation Description

In the Test A setup the pre- and post-change distributions are defined as $f_0 \sim \mathcal{N}(0, 1)$ and $f_1 \sim \mathcal{N}(0.1, 1)$. The parameters for the test are $h = 0$, $\mu = 0.01$, and $d = 10$. The idea of this test is to see how the DE-CUSUM algorithm performs relative to the CUSUM algorithm when looking for a very small mean shift in noisy observations. The generated observations are shown in Fig. 2.1. The change point for this particular test run is denoted by a red dot in the observations. Everything prior to the red dot are observations drawn from distribution f_0, and all the observations following it are drawn from the distribution f_1, illustrating how difficult it is to tell the change has occurred by human evaluation.

In Test B the pre- and post-change distributions are defined as $f_0 \sim \mathcal{N}(0, 1)$ and $f_1 \sim \mathcal{N}(0, 1.1)$. The parameters for the test are $h = 0$, $\mu = 0.001$, and $d = 10$. The idea of this test is to see how the DE-CUSUM algorithm

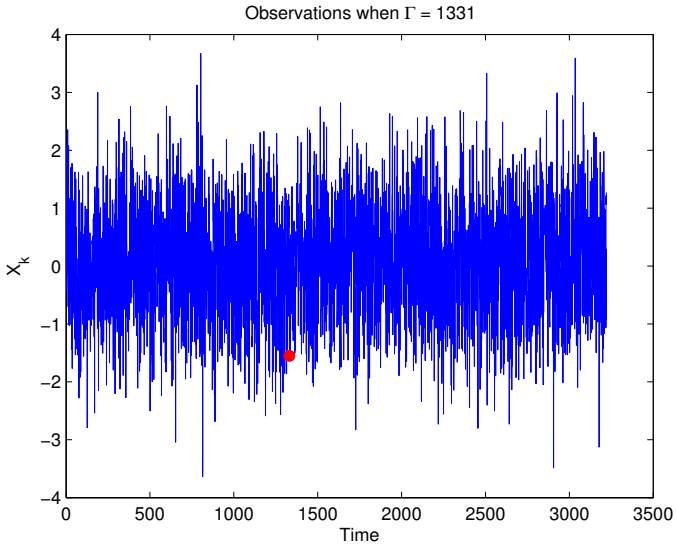


Figure 2.1: X_k Observations for Test A

performs relative to the CUSUM algorithm when looking for a very small change in the standard deviation of the observations. Again the change point is denoted by a red dot in the observations, and we see again how difficult it can be to reliably determine a change has occurred by mere human inspection.

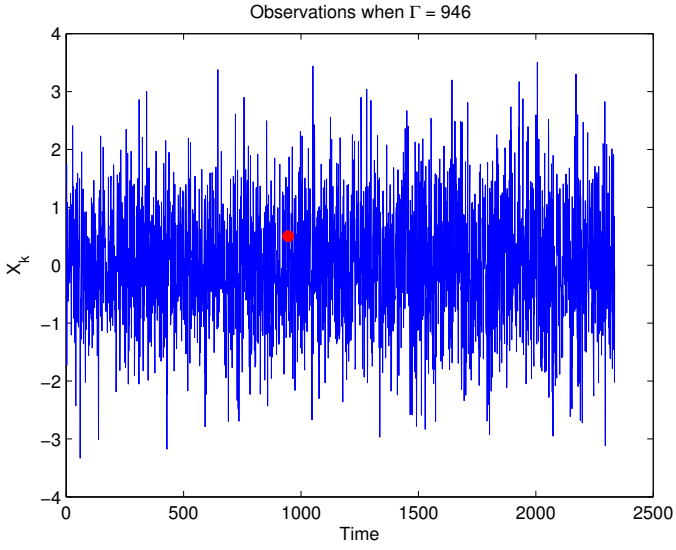


Figure 2.2: X_k Observations for Test B

2.3.2 Numerical Results

In Fig. 2.3 we see the statistic W_k plotted for both the DE-CUSUM and the CUSUM algorithms using the observations shown in Fig. 2.1. The PDC for the DE-CUSUM algorithm in this simulation was 0.4436, so the DE-CUSUM algorithm used a little under half as many measurements as the CUSUM algorithm before the change occurred. The detection delay was 1823 time steps for the CUSUM algorithm, and 1889 time steps for the DE-CUSUM algorithm, so the delay penalty for using DE-CUSUM was 66 time steps, or 3.62% of the CUSUM algorithms total detection delay. In Fig. 2.3 the change point is denoted by a black dot near $k = 1400$.

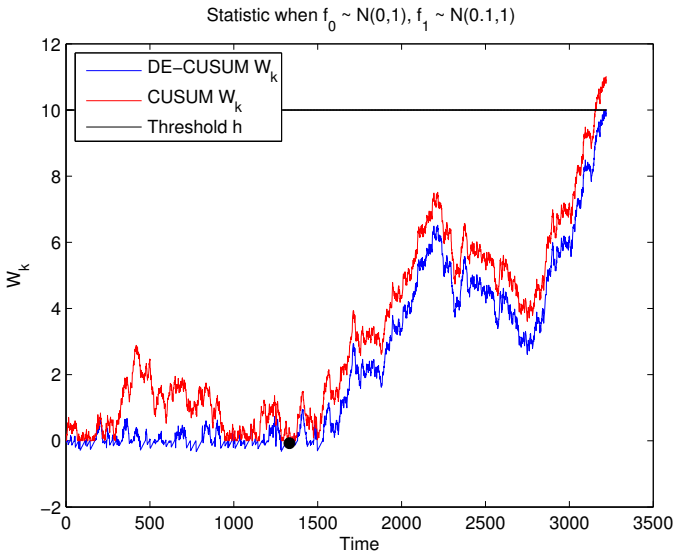


Figure 2.3: W_k Statistic for Test A

Figure 2.4 shows W_k for both DE-CUSUM and the CUSUM algorithms using the observations from Test B shown in Fig. 2.2. The PDC was 0.1153 for this simulation, demonstrating the effect adjusting the μ parameter can have on energy efficiency. The DE-CUSUM algorithm used about one-tenth the number of measurements that CUSUM used before the change occurred. The trade-off can be seen by examining the detection delays for the two algorithms. The detection delay for CUSUM was 1061 time steps, and the delay for DE-CUSUM was 1389 time steps. The penalty was much greater for DE-CUSUM in this test than it was in Test A; it becomes 328 time steps or 30.91% of the CUSUM algorithms total detection delay. Again the black dot near $k = 950$ denotes the change point in the test.

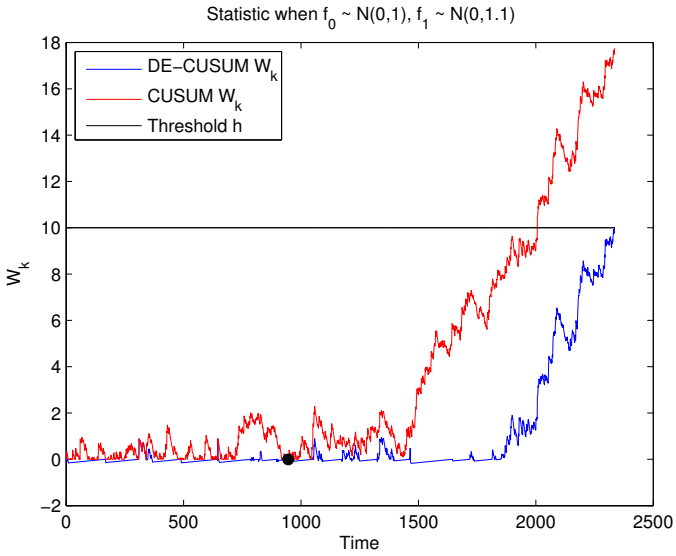
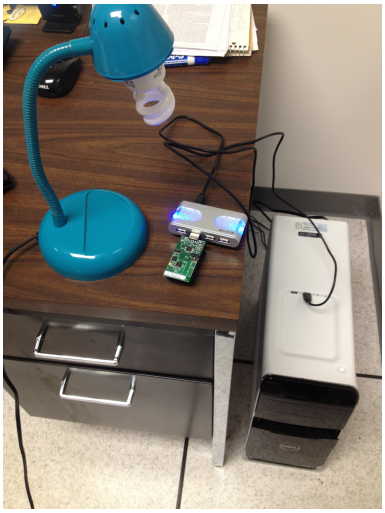


Figure 2.4: W_k Statistic for Test B

2.4 Testbed Results

This section goes over a basic implementation of the DE-CUSUM algorithm on an actual testbed. A picture of this testbed can be seen in Fig. 2.5.



(a) Full Testbed Setup

(b) TelosB Sensor Node

Figure 2.5: DE-CUSUM Testbed

Light was the medium used for detection during the DE-CUSUM implementation testing. This particular choice of medium means the observations X_k have quite a different behavior than the simulations of the algorithm

run previously. With the light the variance is lower, and the mean shift is much higher between the pre- and post-change distributions. When the Kullback-Leibler (K-L) divergence is defined as

$$D_{KL}(f_1||f_0) = \int_{-\infty}^{\infty} \ln\left(\frac{f_1(x)}{f_0(x)}\right) f_1(x)dx,$$

the differences mean that the K-L divergence between f_1 and f_0 is much larger for the testbed than it was in the simulations, and in turn implies that the algorithm will tend to react very strongly to even slight changes in the measured observations X_k. This is seen during the tests in the form of large changes in W_k for even a single time step.

2.4.1 Testbed Description

The testbed developed for the implementation of the DE-CUSUM algorithm utilizes a single TelosB sensor mote device [16] connected to a computer using Matlab. The mote is running a simple C based application on top of the Contiki OS [17]. The mote assumes that the pre- and post-change distributions are i.i.d. Gaussian distributions, and calculates the mean and variance for both by taking samples under both conditions prior to a test run. The idea is to expose the mote to pre-change conditions, then hit the reset button, and then wait until the indicator LED turns off (indicating the samples have been gathered and the mean and variance have been calculated). Next expose the mote to post-change conditions and repeat the process hitting the general purpose button instead of the reset button. Immediately after the indicator light turns off a second time, the mote begins running the DE-CUSUM algorithm.

The mote does not need to be connected to a computer to run the DE-CUSUM algorithm, but it is in the testbed so that data can be dumped to the computer as it comes in for visualization purposes. In both situations when the red LED turns on it indicates that a measurement is being taken, and when the blue LED turns on it indicates that the change has been detected. Instead of immediately ending the test, however, the sensor keeps running and allows the statistic W_k to run indefinitely. This is exclusively because it is a useful feature for demonstration purposes, and in applications the change

detection would probably trigger some sort of alert. The blue LED remains on as long as $W_k > d$.

2.4.2 Numerical Results

The X_k observation values and W_k statistic are shown for Test A in Fig. 2.6. In this test the light was off for the first sample taken, and was switched on around 5 seconds into the test. The observations jump about 150 lux, and continue to rise over the duration of the test. The continued rise results from the lamp heating up, causing its light output to rise slowly. This natural heating up process shows the relationship between the observed X_k values and the sleep times where $W_k < 0$. To get this data the lamp was close to the sensor when the post-change distribution was sampled, and was moved farther away during the test so that the W_k statistic would not immediately rise towards infinity. In the sufficient statistic plot of Fig. 2.6 the blue line is W_k, the red line is just a zero line to show where a sample is taken, and the green line is the d threshold. The point of this test is to show that as the observations become more like f_1 the sensor sleeps less and the PDC rises.

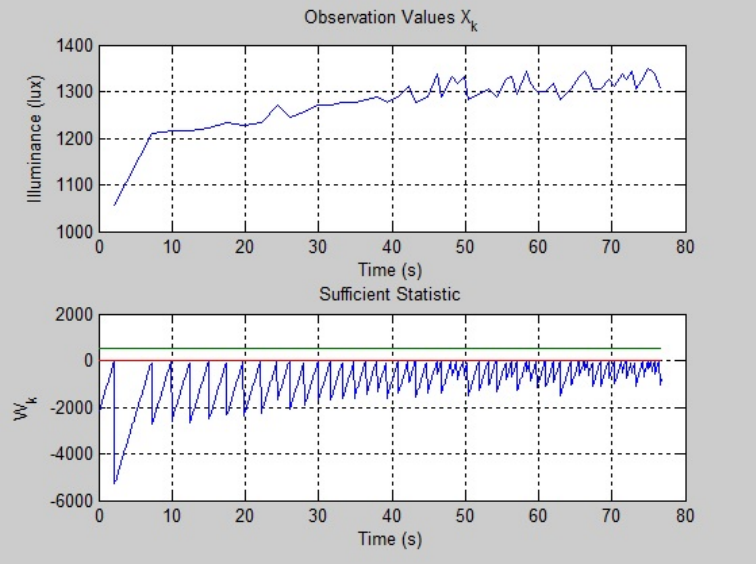


Figure 2.6: DE-CUSUM Implementation Test A

Figure 2.7 depicts the X_k and W_k values for Test B. In this test the lamp was initially off, and was switched on about 5 seconds into the test as well. The lamp is then given until about 60 seconds into the test to warm up

some. After this the lamp is incrementally nudged a little bit closer every 10-20 seconds. Finally at the 109 second mark $W_k > d$ and the test is theoretically over. As mentioned earlier the code does not stop running at this point, and we see W_k continuing to rise even after $W_k > d$. Again, as we see the lamp being moved closer to the sensor we see more X_k observations being taken more often, and the PDC rises.

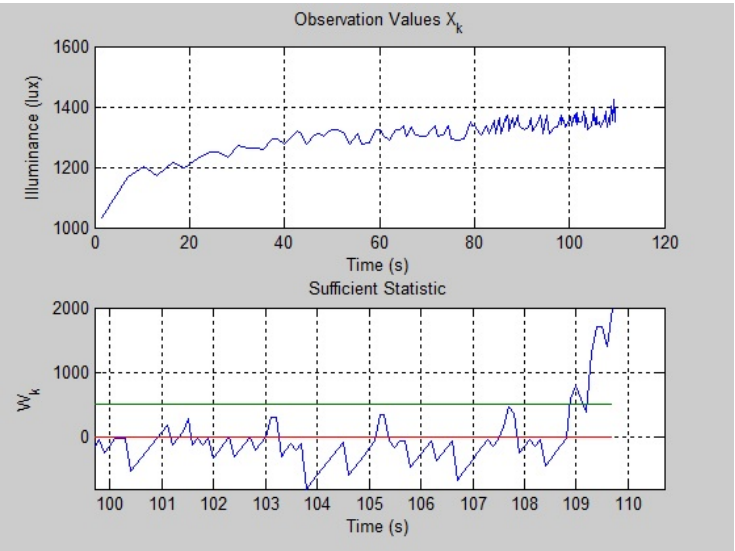


Figure 2.7: DE-CUSUM Implementation Test B

2.5 Conclusion

The implementation of DE-CUSUM on the sensor mote was quite effective, easily showing how even a simplistic approximation of a distribution can have effective results. The Gaussian assumption for the distributions was chosen only for the sake of computational simplicity on the sensor node, but in reality the assumption would work well for many systems, and would work well enough as an approximation for many non-Gaussian systems.

A few interesting ideas for expansion on this work are as follows:

- Implement the testbed with a system using non-Gaussian assumptions.
- Implement the testbed on a 'real' system such as a bridge, or a machine on an assembly line.

- Implement a sensor network extension [7] of the testbed.

These ideas would be useful for generalizing the results obtained in the implemented testbed, and testing the algorithm in an even more applied setting. This work is a very strong candidate for more applied testing; if the applications worked well then the research would have a very solid motivation backing future work, and would most likely benefit from funding provided by potential users of the algorithm as well.

CHAPTER 3

SENSOR SLEEP CONTROL FOR ENERGY EFFICIENT TRACKING IN A WIRELESS SENSOR NETWORK

In this chapter the goal is to develop and implement an algorithm that can track an object moving through a sensor network, using sleep control to maximize energy efficiency in the network. The motivation for energy efficiency in the network is to reduce maintenance costs and increase longevity of the sensors in a network deployed in a particularly unforgiving or harsh environment. Changing the batteries for a large network is a significant investment of time and money, and if the environment is hostile then the safety of the maintenance crews may be of concern as well. Additionally, these intelligent control schemes cost virtually nothing to use in lieu of a scheme such as duty cycling, which does not take into account the sensor observations. This is because the observations are already available to the sensors as part of their function. Basic information theory implies that intelligently using the information they provide for control can only increase the performance of the network; basic duty cycling should be a lower bound on what can be accomplished when using this information.

3.1 Simple Model

In this section a simple model for a tracking WSN is presented as described in [11]. Based on this model the optimization problem of sleep control for the nodes can be framed in such a way that it is possible to approach with dynamic programming. The use of policy iteration as described in [18] can find the optimal solution, but can also be computationally infeasible for a WSN with a significant number of sensors. This provides the motivation to find a sub-optimal solution that is more tractable.

3.1.1 Problem Setup

Consider a discrete time 2-dimensional space in which a single mobile object is being tracked over time. The space is discretized by partitioning it into n enumerated cells, and then representing the object's position with a discrete variable simply designating which cell the object currently resides in. A finite alphabet $\mathcal{B}$ is used to describe the set of all possible locations (cells) the object could be in at any given point in time.

The object's movement is described by a discrete time Markov chain whose states correspond with the cells in the space. The state of the chain at time k is the cell that the object resides in at time k. There is one additional terminal state, $\mathcal{T}$, which represents the case when the object leaves the area of space the network is concerned with. It is assumed that once the object leaves the network space and enters the state $\mathcal{T}$, it never leaves $\mathcal{T}$, and that the central controller is made immediately aware that the system is in $\mathcal{T}$. This means that $|\mathcal{B}| = n + 1$. The movement of the object can be described then by a $(n+1)\times(n+1)$ probability transition matrix P, where each element P_{ij} and $i, j \leq n$ of the matrix represents the probability the object will move from cell i to cell j in the next time step. Note then that as described previously $P_{\mathcal{T}\mathcal{T}} = 1$ and $P_{\mathcal{T}j} = 0, \forall j \in \mathcal{B} - \mathcal{T}$. The location of the object at time k is denoted as b_k. It is assumed that there is a path from every state to $\mathcal{T}$, so that $\lim_{k\to\infty} b_k = \mathcal{T}$. The distribution for b_{k+1} conditioned on b_k is described by

$$b_{k+1} \sim \boldsymbol{e}_{b_k} P \tag{3.1}$$

Next consider the sensor behavior in the network. One sensor is placed in each cell of the network. A sensor can be either active, or sleeping in each time step k. If a sensor l is active at time k it will take a measurement $s_{k,l} \in \{0, 1\}$, with $s_{k,l} = 1$ corresponding to the case where the object is in the cell of sensor l at time k, and with $s_{k,l} = 0$ corresponding to the case where the object is in some other cell at time k. When $s_{k,l}$ is sent to the central controller, the controller then decides how long sensor l should sleep, starting at time $k + 1$. The sleep time remaining for a sensor l at time k is denoted as $r_{k,l}$. If $r_{k,l} > 0$ then sensor l is asleep at time k, and will not take any measurements. The evolution of the sleep times for sensor l can be

described mathematically as

$$r_{k+1,l} = (r_{k,l} - 1)\mathbb{1}_{\{r_{k,l}>0\}} + u_{k,l}\mathbb{1}_{\{r_{k,l}=0\}}, \tag{3.2}$$

where $u_{k,l}$ is the sleep time supplied by the central controller to sensor l at time k. The first summand of the right-hand side represents the sleep timer counting down, and the second summand represents the sensor l getting a new sleep time $u_{k,l}$ from the central controller when it wakes up.

The state of the system at time k is given by $x_k = (b_k, \boldsymbol{r}_k)$. The equations (3.1) and (3.2) describe how the state of the system evolves over time. The state x_k is only partially observed because the system does not always know b_k. The total information available to the central controller at time k is

$$I_k = (\boldsymbol{s}_0..., \boldsymbol{s}_k, \boldsymbol{r}_0, ..., \boldsymbol{r}_k, \boldsymbol{u}_0, ..., \boldsymbol{u}_{k-1}), \tag{3.3}$$

with $I_0 = (\boldsymbol{s}_0, \boldsymbol{r}_0)$ denoting the known initial state of the system. The sleeping policy μ_k is such that

$$\boldsymbol{u}_k = \mu_k(I_k). \tag{3.4}$$

There is an energy cost $c \in (0, 1]$ for each active sensor at each time step, and a tracking cost of 1 for each time step the object is not observed. The cost c will act as a trade-off parameter to adjust how much tracking error the system will tolerate versus how much energy consumption. Once the object enters $\mathcal{T}$ it is assumed that no farther costs are incurred on the system and the problem terminates. The total cost at time k for the system can be written as

$$g(x_k) = \mathbb{1}_{\{b_k \neq \mathcal{T}\}} \left(\mathbb{1}_{\{r_{k,b_k}>0\}} + \sum_{l=1}^{n} c\mathbb{1}_{\{r_{k,l}=0\}} \right). \tag{3.5}$$

The total cost for the system is given by

$$J(I_0, \mu_0, \mu_1, ...) = \mathbb{E}\left[\sum_{k=1}^{\infty} g(x_k) \middle| I_0 \right]. \tag{3.6}$$

This function is well defined because g is clearly bounded by $(cn+1)$ and the expected time for the object to leave the network is finite. The optimization

problem is then to compute

$$J^*(I_0) = \min_{\mu_0,\mu_1,\ldots} J(I_0, \mu_0, \mu_1, \ldots). \tag{3.7}$$

The solution to this for every value of c gives an optimal sleeping policy. This type of problem lies in the framework of a partially observable Markov decision process (POMDP).

3.1.2 Using a Sufficient Statistic

The formulation from the previous section is a good start, but there is a large problem in that the information given in (3.3) is unbounded in memory. To fix this a sufficient statistic is needed that is bounded in memory. In the POMDP field the sufficient statistic given by the probability distribution of the state x_k given I_k is known as the belief state [18], [19], [20]. The belief state can be written as $v_k = (\boldsymbol{p}_k, \boldsymbol{r}_k)$, with $\boldsymbol{p}_k$ a row vector of length $n+1$ that denotes the probability mass function of b_k given I_k.

$$\boldsymbol{p}_{k,l} = \mathbb{P}(b_k = l | I_k). \tag{3.8}$$

Vector $\boldsymbol{p}_k$ evolves according to

$$\boldsymbol{p}_{k+1} = \boldsymbol{e}_{\mathcal{T}} \mathbb{1}_{\{b_{k+1}=\mathcal{T}\}} + \boldsymbol{e}_{b_{k+1}} \mathbb{1}_{\left\{r_{k+1,b_{k+1}}=0\right\}} + [\boldsymbol{p}_k P]_{j:r_{k+1,j}>0} \mathbb{1}_{\left\{r_{k+1,b_{k+1}}>0\right\}}, \tag{3.9}$$

where b_{k+1} (conditioned on $\boldsymbol{p}_k$) is distributed as

$$b_{k+1} \sim \boldsymbol{p}_k P. \tag{3.10}$$

From here the policy and cost functions can be rewritten in terms of v_k. The sleeping policy defined in (3.4) becomes

$$\boldsymbol{u}_k = \mu_k(v_k). \tag{3.11}$$

The total cost defined in (3.6) becomes

$$J(v_0, \mu_0, \mu_1, \ldots) = \mathbb{E}\left[\sum_{k+1}^{\infty} g(x_k) \middle| v_0\right]. \tag{3.12}$$

The optimal cost defined in (3.7) becomes

$$J^*(v_0) = \min_{\mu_0,\mu_1,\ldots} J(v_0, \mu_0, \mu_1, \ldots). \tag{3.13}$$

3.1.3 Optimal Solution Using Dynamic Programming

Now that the optimization problem has been phrased in terms of a convenient sufficient statistic v_k, dynamic programming can be applied to find a solution. Given that the Markov chain is stationary it can be shown (e.g. see [18]) that there exists a stationary optimal policy μ^* for the problem. This policy and the optimal cost J^* can be found by solving the Bellman equation

$$J(v) = \min_{\mu} \mathbb{E}\left[g(x_1) + J(v_1) | v_0 = v, \boldsymbol{u}_0 = \mu(v)\right], \tag{3.14}$$

with μ^* the minimizer for μ. An analytical solution to (3.14) is too difficult to find, so the use of an iterative technique must be employed such as value or policy iteration [18]. Even through the use of these methods, however, the curse of dimensionality causes this to be an intractable problem, and a sub-optimal solution remains the only feasible option.

3.1.4 Sub-Optimal Solutions

In the first cost reduction (FCR) solution the assumption is made that there will be no future observations at each time step. This means that the belief state is expected to evolve according to

$$\boldsymbol{p}_{k+1} = \boldsymbol{p}_k P \tag{3.15}$$

rather than as in (3.9). There will still be future observations that enable the use of Bayesian statistics; it is simply that when determining a sensor's future sleep policy it is assumed that these observations will not be made. The cost at sensor l then becomes

$$J^{(l)}(\boldsymbol{p}) = \min_{u} \left(\sum_{j=1}^{u} \left[\boldsymbol{p}P^j\right]_l + \sum_{i=1}^{n} c \left[\boldsymbol{p}P^{u+1}\right]_i + J^{(l)}\left(\boldsymbol{p}P^{u+1}\right) \right), \tag{3.16}$$

where the first term represents the tracking cost, the second the energy cost, and the third the future cost. This makes (3.16) a Bellman equation for the per-sensor problem under FCR assumptions. The solution to this equation is given as follows:

$$J^{*(l)}(\boldsymbol{p}) = \sum_{j=1}^{\infty} \min \left\{ \left[\boldsymbol{p}P^j\right]_l, \sum_{i=1}^{n} c \left[\boldsymbol{p}P^j\right]_i \right\}. \tag{3.17}$$

If the set $\mathcal{U}(\boldsymbol{p})$ is defined as

$$\mathcal{U}(\boldsymbol{p}) = \left\{ u : \left[\boldsymbol{p}P^{u+1}\right]_l \geq \sum_{i=1}^{n} c \left[\boldsymbol{p}P^{u+1}\right]_i \right\}, \tag{3.18}$$

then the per-sensor policy $\mu^{*(l)}(\boldsymbol{p})$ is

$$\mu^{*(l)}(\boldsymbol{p}) = \min_{u \in \mathcal{U}(\boldsymbol{p})} u, \tag{3.19}$$

which simply means the policy at sensor l is to wake up the first time the expected tracking costs are larger than the expected energy cost.

A $\mathrm{Q_{MDP}}$ solution is one where it is assumed the partially observed state becomes fully known after a control input has been selected (e.g. see [19], [20]). This replaces (3.9) with

$$\boldsymbol{p}_{k+1} = \boldsymbol{e}_{b_{k+1}} \tag{3.20}$$

for the belief state evolution. Then the Bellman equation for sensor l under the assumptions made becomes

$$J^l(\boldsymbol{p}) = \min_u \left(\sum_{j=1}^{u} \left[\boldsymbol{p}P^j\right]_l + \sum_{i=1}^{n} c \left[\boldsymbol{p}P^{u+1}\right]_i + \sum_{i=1}^{n} \left[\boldsymbol{p}P^{u+1}\right]_i J^{(l)}(\boldsymbol{e}_i) \right) \tag{3.21}$$

and can be solved to find the cost function and the policy for sensor l. The first summand on the right-hand side of the equation is the tracking cost, the second summand is the energy cost, and the third summand is the future cost. An analytical solution to (3.21) is not easily found, but if a solution is found for $\boldsymbol{p} = \boldsymbol{e}_b, \forall b \in \{1, .., n\}$, then a solution for all other values of $\boldsymbol{p}$ can be found. The use of policy iteration [18] will yield the solutions for $b \in \{1, .., n\}$, and the farther details of this process can be found in [11].

The Q_{MDP} simplifying assumption actually assumes more information than is available to the network at time k, so the cost function found through the Q_{MDP} assumption acts as a lower bound for the optimal performance of the network.

3.2 General Model

In this section a more general model for an energy efficient tracking WSN is developed, as given in [13]. Again, the optimization problem of sleep control for the sensor nodes can be framed in a way such that dynamic programming can be applied to solve the problem. The primary difference between this more general model and the simpler one from the previous section is that the sensors positions can now be anywhere in the continuous space, rather than only at discrete points in the space. Additionally the observations become a measurement of the medium of concern (e.g. RSSI, temperature, light or sound intensity) rather than a binary measurement indicating whether the object is in a particular cell or not. These generalities better model a real system than the setup used in the simplified model.

3.2.1 Problem Setup

The problem setup for the more general model builds upon the simplified setup presented earlier, and therefore is very similar in many respects. The partitioned space where the object resides in a particular cell remains the same, and $\mathcal{B}$ represents the finite alphabet describing the set of all the possible locations (cells) the object could be in at time k. The movement model for the object is identical to the simplified problem setup.

The n sensors now can exist anywhere in the 2-dimensional network space, regardless of how the space is partitioned into cells for $\mathcal{B}$, the set of possible object locations. At each time k a sensor l can still be either active or asleep. When awake the sensor takes a measurement, but it costs more energy to do so, whereas when asleep the sensor takes no measurement, but uses no energy. At each time step a central controller tells each active sensor how long to sleep before waking up again, and the sensor's sleep cannot be preempted and awoken before its sleep timer has expired. The equation (3.2) governing

the evolution of the sleep times of the sensors still holds as it did in the simple model, with the variables all defined as they were in the simple model.

Given this discrete-time dynamic model the state of the system at time k is denoted by $x_k = (b_k, \boldsymbol{r}_k)$. The central controller does not know the location of the object, b_k though, so the system has only partially observed state information.

The observations for the model are written as

$$z_k = (\boldsymbol{s}_k, \boldsymbol{r}_k),$$

where $\boldsymbol{s}_k$ is a $(n+1)$-vector of observations drawn from a probability measure $\sigma_{\boldsymbol{x}_k}$ which depends on x_k. If a sensor is not awake at time k then its observation is an erasure, and $s_{k,n+1}$ is always a binary observation that simply indicates whether the object has left the network, and the test is over.

The information available to the central controller at time k is

$$I_k = (z_0, ..., z_k, \boldsymbol{u}_0, ..., \boldsymbol{u}_{k-1}),$$

with $I_0 = z_0$ giving the known initial state of the system. The control input for the sensors at time k is given by

$$\boldsymbol{u}_k = \mu_k(I_k).$$

The vector-valued function μ_k is the sleeping policy for the sensors at time k.

The energy cost is still denoted as c, and is given at each time step by

$$\sum_{l=1}^{n} c\mathbb{1}_{\{r_{k,l}=0\}}.$$

The tracking cost is defined in terms of the estimated object location at time k, denoted by $\hat{b}_k$, and the actual object location at time k, b_k. The tracking cost is some distance measure

$$d(b_k, \hat{b}_k).$$

For the purposes of this thesis the squared Euclidean distance will be used

as the distance measure

$$d(b_k, \hat{b}_k) = ||\hat{b}_k - b_k||_2^2$$

Then parameter c trades off energy consumption and tracking error. The optimal choice of $\hat{b}_k$ can be written as

$$\beta_k^*(I_k) = \arg\min_{\hat{b}} \mathbb{E}\left[d(b_k, \hat{b}_k) I_k\right].$$

The total cost for time step k is then

$$g(b_k, I_k) = \mathbb{1}_{\{b_k \neq \mathcal{T}\}} \times \left(d(b_k, \beta_k^*(I_k)) + \sum_{l=1}^{n} c\mathbb{1}_{\{r_{k,l}=0\}}\right),$$

where $\mathcal{T}$ is defined as the terminal state, as in the simplified model. The infinite horizon cost for the system is

$$J(I_0, \mu_0, \mu_1, ...) = \mathbb{E}\left[\sum_{k=1}^{\infty} g(b_k, I_k) \middle| I_0\right].$$

The goal is to find the minimal cost over all possible control policies at each point in time, so

$$J^*(I_0) = \min_{\mu_0, \mu_1, ...} J(I_0, \mu_0, \mu_1, ...). \tag{3.22}$$

For each value of c the solution of (3.22) gives an optimal sleeping policy. The problem is a POMDP, as in the simple setup.

Through the use of a sufficient statistic again the problem can be made so that the information needed at each time step is not unbounded in memory. A belief state is once again used to solve this issue. The sufficient statistic can be written as $v_k = (p_k, \boldsymbol{r}_k)$, where p_k is a probability measure on $\mathcal{B}$ so that

$$p_k(\mathcal{X}) = \mathbb{P}(b_k \in \mathcal{X} | I_k).$$

The prediction p_{k+1} can be computed using standard Bayesian techniques as the posterior measure resulting from prior pP and the observations $\boldsymbol{s}_{k+1}$.

Now the function β_k^* used to find $\hat{b}_k$ can be written using p_k and $\boldsymbol{r}_k$ instead of I_k as

$$\beta_k^*(p_k, \boldsymbol{r}_k) = \arg\min_{\hat{b}} \int_{\mathcal{B}} d(b_k, \hat{b}) p_k(db).$$

Since the state evolution is stationary, β_k^* can be referred to as β^*, which is a function of only p_k. The cost can now be written as

$$\begin{aligned} g(p_k, \boldsymbol{r}_k) &= \int_{\mathcal{B}} \mathbb{1}_{\{b \neq \mathcal{T}\}} \left(d(b, \beta^*(p_k)) + \sum_{l=1}^{n} c\mathbb{1}_{\{r_{k,l}=0\}} \right) p_k(db) \\ &= \int_{\mathcal{B}-\mathcal{T}} \left(d(b, \beta^*(p_k)) + \sum_{l=1}^{n} c\mathbb{1}_{\{r_{k,l}=0\}} \right) p_k(db). \end{aligned} \tag{3.23}$$

The selection of sleep times can be rewritten as

$$\boldsymbol{u}_k = \mu_k(p_k, \boldsymbol{r}_k).$$

The infinite horizon cost becomes

$$J(p_0, \boldsymbol{r}_0, \mu_0, \mu_1, ...) = \mathbb{E}\left[\sum_{k=1}^{\infty} g(p_k, \boldsymbol{r}_k) \middle| v_0 \right]$$

and the optimal cost (3.22) becomes

$$J^*(p_0, \boldsymbol{r}_0) = \min_{\mu_0, \mu 1, ...} J(p_0, \boldsymbol{r}_0, \mu_0, \mu_1, ...). \tag{3.24}$$

3.2.2 Sub-Optimal Solutions

The optimal problem was abandoned in the simple setup, and there is no reason to believe it would be any easier to solve in a more complicated general setup, so sub-optimal solutions are immediately sought after. Intuition from the solution to the simplified setup will be employed to make the solution to the general problem more tractable. The problem will be rewritten as a number of subproblems, each having variable tracking cost expressions that will have to be determined through Monte Carlo simulations gathered prior to beginning to track, or through the use of data gathered while tracking. The idea is that if the tracking cost expressions generated capture the typical behavior of the actual tracking cost then the sleeping policies generated should perform well.

First the problem is broken down into l subproblems, one for each sensor. $J^{(l)}$ is the cost function for the lth sensor approximate subproblem. $T^{\Delta}(b, l)$ is the increase in the tracking cost when sensor l does not wake up at time k

given that the last known location of the object is b. This variable represents how much the lth sensor contributes to the total tracking cost, i.e. how important sensor l is in determining the objects location.

As used before, the $\mathrm{Q}_{\mathrm{MDP}}$ solution to a POMDP problem assumes that the system will be perfectly observable after the current control [20], [21], so

$$p_{k+1} = \delta_{b_{k+1}}.$$

Now a $\mathrm{Q}_{\mathrm{MDP}}$ per-sensor Bellman equation can be developed as in the simplified model

$$J^{(l)}(p) = \min_u \left(\sum_{j=0}^{u-1} \int_{\mathcal{B}-\mathcal{T}} T^{\Delta}(b,l)(pP^j)(db) + \int_{\mathcal{B}-\mathcal{T}} \left(c + J^{(l)}(\delta_b)\right)(pP^{u+1})(db) \right), \quad (3.25)$$

where the summation term is the expected tracking cost of sleeping sensor l for u time units. The second term is composed of the energy cost of coming awake after $u+1$ time units and the cost to go with the observable after control assumption. Given a function for T^{Δ}, and a finite $\mathcal{B}$, the problem can be solved through standard policy iteration [18].

Also as before an FCR solution can be found as well, where it is assumed that there will be no future observations after control, so

$$p_{k+1} = p_k P.$$

The per-sensor Bellman equation defined as

$$J^{(l)}(p) = \min_u \left(\sum_{j=0}^{u-1} \int_{\mathcal{B}-\mathcal{T}} T^{\Delta}(b,l)(pP^j)(db) + c \int_{\mathcal{B}-\mathcal{T}} (pP^{u+1})(db) + J^{(l)}(pP^{u+1}) \right). \quad (3.26)$$

Given a function for T^{Δ}, the solution to (3.26) is

$$J^{(l)}(p) = \sum_{j=0}^{\infty} \min \left\{ \int_{\mathcal{B}-\mathcal{T}} T^{\Delta}(b,l)(pP^j)(db), c \int_{\mathcal{B}-\mathcal{T}} (pP^{j+1})(db) \right\}, \quad (3.27)$$

so the policy is to choose the first u such that

$$c \int_{\mathcal{B}-\mathcal{T}} (pP^{u+1})(db) \geq \int_{\mathcal{B}-\mathcal{T}} T^{\Delta}(b,l)(pP^{u})(db). \tag{3.28}$$

This means the policy is to come awake the first time that the expected tracking cost is greater than or equal to the expected energy cost.

Unlike in the simplified model, the $\mathrm{Q}_{\mathrm{MDP}}$ solution does not represent a lower bound on the optimal value function. What remains is to find an appropriate method of determining values for $T^{\Delta}(b,l), \forall b \neq \mathcal{T}, \forall l$.

Assuming $\mathcal{B}$ is a finite space, and $b_{k-1} = b$, then to generate $T^{\Delta}(b,l)$ for a particular l an assumption about the 'baseline' behavior for the sensors is made. This means an assumption about the set of sensors that are awake at time k is made given $b_{k-1} = b$. Either all sensors are asleep, or the set of awake sensors is selected through a greedy algorithm. For the greedy algorithm the sensor that causes the largest decrease in the expected tracking cost is added to the awake set of sensors until any additional reductions due to a single sensor amount to less than c. To avoid numerical integration the expected tracking cost can be evaluated through Monte Carlo simulation, which simply means simulating the system repeatedly from time $k-1$ to time k.

Given this set of awake sensors, $T^{\Delta}(b,l)$ is computed as the absolute difference in expected tracking cost incurred by changing sensor l from the asleep to the awake state. This can be found again through the use of Monte Carlo simulation. This procedure can be thought of as linearizing the tracking cost about a baseline behavior.

A basic learning algorithm can also be applied to update the values for $T^{\Delta}(b,l)$ as the test goes on, based on the measurements the sensors are making. For the sake of brevity the description of this algorithm is left out, and the details can be found in section III.C of [13].

In the remaining sections there will be references to the T^{Δ} matrix, which is simply the values of $T^{\Delta}(b,l)$ arranged into a matrix where b and l are the index terms of the columns and rows of the matrix, respectively.

3.3 Intuitive Solution

The results obtained by the authors of [13] are quite powerful, and give very near approximations for the optimal solution. Unfortunately, the computational complexity of their solutions can be prohibitive in an actual implementation of a WSN with current technology. In particular, the construction of the $T^{\Delta}(b, l)$ matrix in [13] is an off-line calculation that will often require a huge number of real samples from the sensors for any implemented network before the algorithm can be implemented. For small networks this can probably be done in a somewhat reasonable amount of time, but for a large WSN this will quickly become far too time-consuming. It takes a fairly long time in simulations of large networks, but it would take at least an order of magnitude greater time when taking real samples due to the added communication delay between the sensors necessary to take the samples.

These complications motivate a different approach to the problem, one whose focus is more on the implementation of the algorithm on a real WSN, and which may not be as optimal as the methods described previously. This approach should be easy to run on-line, and should require minimal, if any, preprocessing. It will avoid some of the obfuscations that arise through the use of tools such as dynamic programming, making it very intuitive. It will be easy to understand, and easy to tweak as necessary for different intended applications.

3.3.1 Problem Setup

The problem setup will be similar to the one posed in [13]. It is assumed there is a single object moving through a network with its movement modeled as a Markov chain, and whose movement statistics can be described as a uniform random walk. Specifically, the object will move to one of its adjacent cells or remain in its current cell all with equal probability. A symmetric structure to the cells is assumed (e.g. tessellated regular polygons in 2-dimensional space) so that this movement can be general and consistent with reality.

The sensors behave as they did previously, having two states of operation: sleeping and awake. When awake there is an energy cost, and when asleep they do not take any samples. The sensors cannot be awoken from their sleep, and will continue to sleep until their sleep timers expire. The sleep

timers are set by the central controller every time the sensor comes awake and delivers an observation.

The observation model used is a rotationally symmetric bivariate Gaussian surface, whose coordinates correspond to the x and y coordinates of the cells, and whose mean is centered on the objects current location. This assumption is actually one of the most restrictive used in the model because it requires the application to be in a relatively open space such as a field, or in the open atmosphere. The observation model for a WSN in a building, for example, would not be expected to be symmetric because physical structures such as walls would obstruct observations between cells in the space.

3.3.2 Use of Intuition

The proposal is to abandon the notion of solving the Bellman equation in the interest of computational simplicity, and use some other method to get a different suboptimal solution that still performs adequately.

Intuition can be gained from looking into the characteristics a good solution should exhibit by examining the behavior of the model in [13]. When plotting $T^{\Delta}(b, l)$ for a fixed object location b at the end of a long test, it is obvious that the values of sensor importance are heavily correlated with the distance from the object. An example of $T^{\Delta}(b, l)$ with b fixed can be seen in Fig. 3.1. In this example the sensors in the network are arranged in an 11x11 rectangular grid.

The movement model assumes the object can move to any adjacent state or remain where it is in a particular time interval. Additionally, the sensor network and the state space are laid out to preserve geometric symmetry. These assumptions allow for a symmetric movement model behavior, which can be exploited in developing a suboptimal solution. All the measurements of the sensors are assumed to be only correlated to the distance from the object, and have no directional component. Given the symmetric model and the non-directional measurements it is assumed that the importance of a particular sensor i to detection of b can be approximated well by an expression that is a function only of d_i, the distance from the sensor i to the object.

The sensors that have the highest importance in detection of the objects

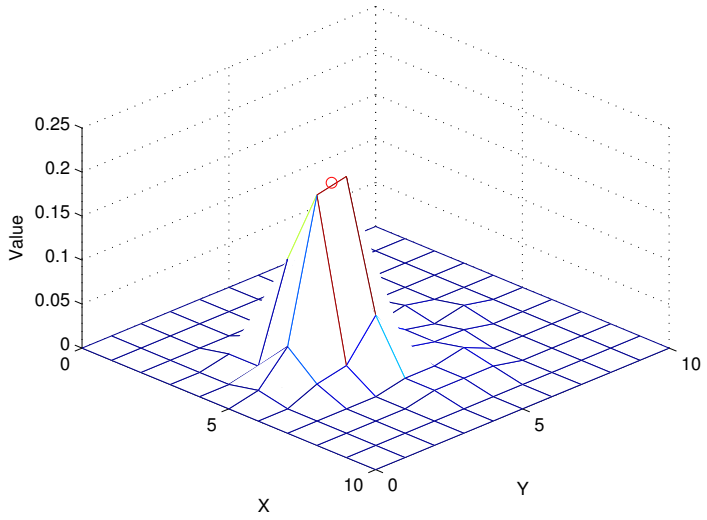


Figure 3.1: $T^{\Delta}(b, l)$ for a Fixed b

location at any particular time k should be the sensors that are awake at time k. Given that this importance value is based only on d_i, as is assumed, the sleeping patterns should be designed such that the sensor i awakens as it becomes more likely that the object is nearby. A given awake sensor i at time k should then sleep for some function of $\mathbb{E}[\tau(d_i)]$, the expected time it would take for the object to travel the distance d_i. $\mathbb{E}[\tau(d_i)]$ will be stationary if it is assumed that the object is on an infinite plane, and so will never leave the network. Although this assumption conflicts with what is known to be the truth, it is made regardless for the sake of simplicity.

3.3.3 Monte Carlo Simulations

The expression $\mathbb{E}[\tau(d)]$ is estimated by setting up a temporary hexagonal shaped network of hexagonal cells that has distance d from its center to any of the edges that define the limit of the network. The object is placed into the middle of this network and a uniform random walk P transition matrix is defined. The length of time the object stays in the network is how long it took the object to move distance d away from its starting point. By running this simulation many times and taking an average of the results, an estimate is obtained for $\mathbb{E}[\tau(d)]$ for a particular value of d. After doing this for a number of different values for d, a function $f(d)$ can be fit to the results

mapping d to $\mathbb{E}[\tau(d)]$. After running this test Fig. 3.2 was created showing

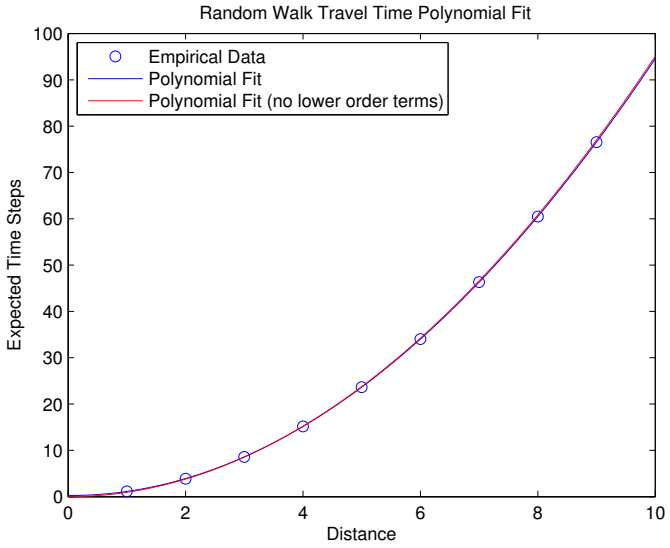


Figure 3.2: $f(d)$ and $\mathbb{E}[\tau(d)]$

the empirical results from the simulations, and a fitted function $f(d)$. It turns out that $f(d)$ is very well approximated by a simple quadratic without lower order term corrections.

Using the given $\hat{b}$, an estimate of the object location, $\hat{d}$ an estimate of the distance from the sensor to the object location can be found. Estimate $\hat{d}$ can then be used to assign a sleep time for any sensor l in the network. It is said that

$$u_{k,l} = af(\hat{d}), \tag{3.29}$$

where $u_{k,l}$ is the amount of time that sensor l should sleep starting at time step $k+1$, and a is some constant. This allows for a to act as a trade-off parameter that indirectly controls how many sensors are active during any given time step, similar to how c functioned in the model presented in [13].

3.4 Simulation Results

The general model is used for the problem setup for all the simulations. The observations are such that $s_{k,l} \in \mathbb{R}$. The observation model assumes that the observation intensity drops off at a rate roughly proportional to the inverse square of the distance $d(b_k, l)$ between the location of the object b_k and the

sensor l making the observation.

$$s_{k,l} = \frac{10}{(d(b_k, l)^2 + 1)} + Z_k \tag{3.30}$$

with noise term Z_k an additive Gaussian random variable distributed as $Z_k \sim \mathcal{N}(0, 1)$. This model is roughly valid for a number of physical applications (e.g. light propagation, sound propagation), making it a nice general observation model to work with.

3.4.1 1-Dimensional Sensor Array

The first network setup analyzed through simulations is a 1-dimensional network with the sensors and states simply arrayed into a straight line. There are 11 sensors arrayed equidistant from each other, and the 21 states are on top of and interleaved between the sensors. Figure 3.3 depicts this configuration.

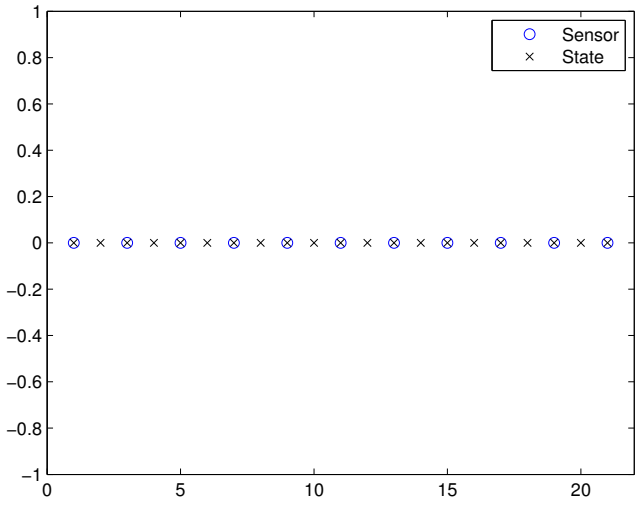
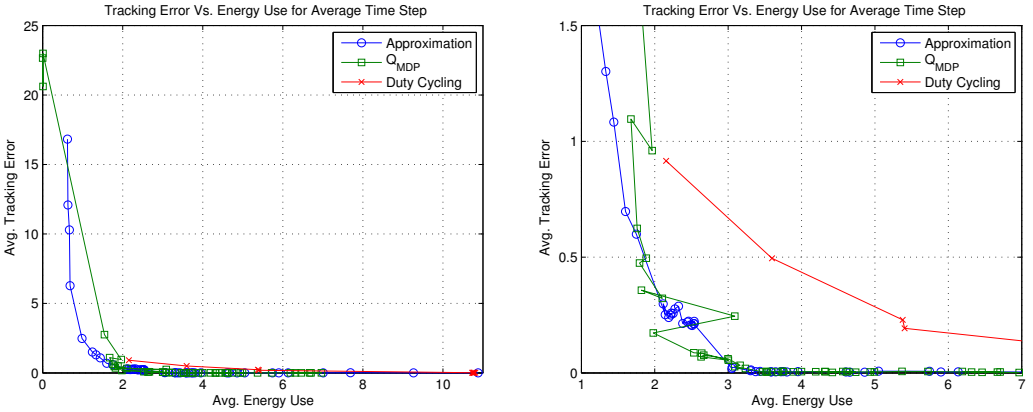


Figure 3.3: 1-Dimensional Array Network Setup

The movement model for the object is a random walk behavior over the state array. The object starts in the middle state of the network, and can either stay where it is, or move to either its left or right neighbors all with equal probability at each time step. It can only leave the network by getting to either the first or the last state of the network at the tips of the line. At

either of these points it has probability $\frac{1}{3}$ to leave the network, ending the test.

The $T^{\Delta}(b,l)$ matrix for the Q_{MDP} solution from [13] was generated using 200 Monte Carlo simulations. After the $T^{\Delta}(b,l)$ matrix generation, 100 simulation runs were performed without contributing to the results for the sole purpose of allowing the learning algorithm to adjust the $T^{\Delta}(b,l)$ matrix. The learning updates used a constant step size of 0.01. Figure 3.4 shows the results of the simulations. Each of the points on these figures is the average of 50 simulation runs for a particular energy cost value, c. The x axis is the average number of sensors awake per time step, and the y axis is the average tracking error per time step. Figure 3.4 was generated by varying c over an average energy usage range of interest.



(a) Zoomed Out (b) Zoomed In

Figure 3.4: 1-Dimensional Array Network Results

First it is instructive to consider what *should* happen ideally in this network setup. The network is a very simple one, just a 1-dimensional line that the object can move along. This combined with the simple movement model employed makes it seem obvious that the best sensors to turn on at every time step are the ones adjacent to the estimated state of the objects last location. Given that the object can not move more than one state per time step, and that the last estimated location was reasonably accurate, it makes sense not to turn on sensors very far from the object's last estimated position. So if the energy use was limited to two active sensors per time step, and if the sensors could be woken up on an as needed basis (which is not the case in the problem setup), the sensors that should be active every time step would be

the ones adjacent to the last object position. This ends up being more or less the behavior seen by the intelligent methods employed in the simulations.

It can be seen from Figure 3.4 that both the Q_{MDP} and the simple approximation methods worked noticeably better than duty cycling the sensors. What is less clear in this simulation is which of the two more intelligent methods worked better. When more than about 3 sensors are on per time step, both methods seem to make no mistakes. When looking just under this threshold, however, it can be seen that the Q_{MDP} method does seem to perform a bit better, disregarding some aberrant data points.

An example of the final T^{Δ} matrix is shown in Fig. 3.5, and it follows what intuition would suggest. The x axis labels the sensors, and the y axis labels the states, so it can be seen that the sensors with the smallest distance to the corresponding states are the most important. This is exactly the intuition exploited in the intuitive approximation developed in this thesis, and that explains why the performance is so similar between the two methods. The intuitive approximation method just has the advantage of not needing to go through the computationally tedious operation of policy iteration to solve the dynamic programming solution derived in [13].

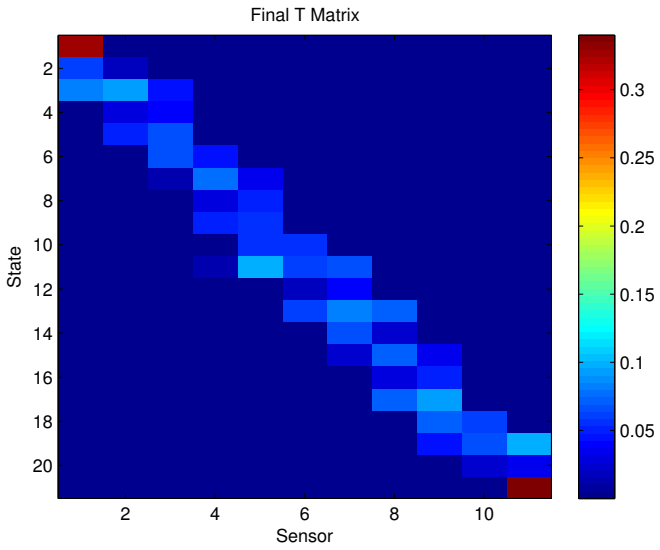


Figure 3.5: Final T^{Δ} Matrix for 1-Dimensional Array Network with $c = 0.01$

While this simulation setup perfectly displays the advantages of the intuitive approximation method, the network itself is rather simple. A more interesting extension is to look at what happens in a 2-dimensional network

space, and this is what is analyzed in the next section.

3.4.2 2-Dimensional Sensor Grid Array

The network setup used in this simulation was a 2-dimensional state space partitioned into tessellated hexagonal cells. The sensors were uniformly arrayed into a 5×5 rectangular grid arranged to fall exactly inside the centers of the cells. There is one sensor in the network for every four states, implying there are 100 states in the setup. This arrangement can be seen in Fig. 3.6.

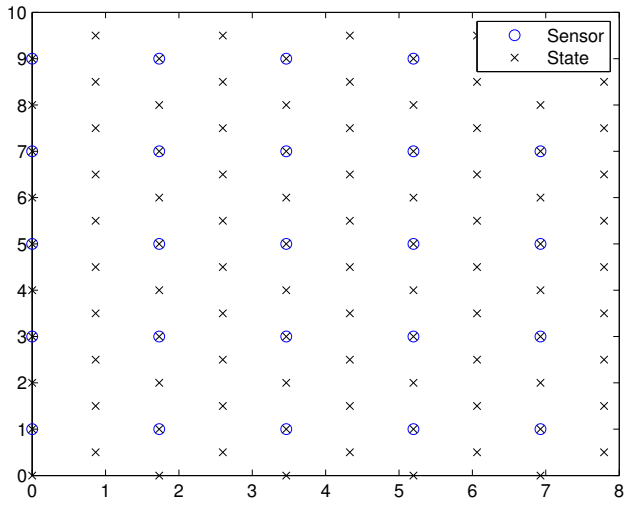


Figure 3.6: 2-Dimensional Grid Network Setup

The simulation assumes the object follows a random walk movement model behavior, similar to the last simulation example. Also the object will again start in the center of the network state space. Now the geometry of the state space is different, however, so there is a uniform $\frac{1}{7}$ probability the object moves to any particular adjacent cell, or remains where it is. If the object is on the right or left edge of the network it has probability $\frac{2}{7}$ to leave the network. If it is on the top or bottom edge is has probability $\frac{1}{7}$ or $\frac{3}{7}$ to leave the network, depending on which column the state is in. If the object is in the upper left or bottom right corners it has probability $\frac{3}{7}$ to leave the network, and if it is in the other corners it has probability $\frac{4}{7}$ to leave.

Each point of the results presented in Figure 3.7 is the average value of 1000 simulations run for a particular c value. The Q_{MDP} per sensor solution

from [13] is presented alongside the approximation result. The duty cycling result was plotted to establish a baseline performance to illustrate the benefit of using intelligent control with the sensors.

As before, it is worth considering how the system *should* behave before analyzing how it did behave. The principles for this network are similar to the last one, only now it is a 2-dimensional space. This means that where two sensors were sufficient to keep a very good track of the object in the last network, this network will require a number closer to three, or more. There is far more uncertainty in the 2-dimensional random walk movement model, and so there is now more incentive to check sensors farther away from the last estimated state of the object's position. This being said on the larger scale, it is still expected that the farther away sensors should not be turned on as much as the closer ones. The intuition will again be mirrored in the observed results of the simulation.

Figure 3.7 shows roughly what was predicted by intuition. For a low number of sensors active per time step any method of tracking is about equally effective, because all methods are almost equally bad. This makes sense considering the random walk movement model assumption, and lack of direction component measurements available from the sensors. Once the number of sensors active per time step is raised enough to actually track the object, gains are seen in tracking accuracy when using intelligent methods of sensor sleeping over duty cycling. Note that the intuitive approximation method proposed in this thesis again performs nearly as well as the $\mathrm{Q_{MDP}}$ per sensor method from [13].

An example of the final T^{Δ} matrix is shown in Fig. 3.8. It is more difficult to analyze what is happening in this example compared to Fig. 3.5, but it is still readable, and actually matches again what would be expected. The sensors closest to the states are the ones with the highest weights, just as before, but now there are as many as three sensors with higher values for a given state because of the 2-dimensional network state space. This T^{Δ} matrix structure follows what the intuitive approximation assumes, so again a very similar performance between the two methods is seen, with the $\mathrm{Q_{MDP}}$ method pulling slightly ahead for some values of c.

This simulation shows that the proposed intuitive approximation method still performs about as well as the $\mathrm{Q_{MDP}}$ solution even in a 2-dimensional network state space. This greatly extends the possible use of the network

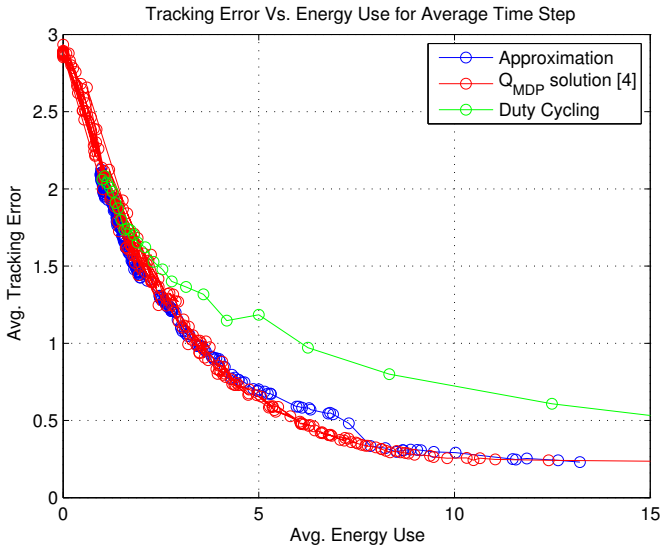


Figure 3.7: 2-Dimensional Grid Network Results

to a larger number of potential applications, as most ground based networks can be roughly modeled as a 2-dimensional environment. Combining these networks could presumably work in the context of a building as well. The hallways could be covered by 1-dimensional arrays, and the larger open areas could be covered by 2-dimensional grid arrays of sensors. The only trick is that the P matrix would have to be carefully designed, and the $\hat{b}$ calculation should only use the sensors in the relevant sub-network that the object is in. Some sort of decision threshold would likely have to be designed for the connection between two sub-networks.

3.4.3 2-Dimensional Sensor Randomized Array

The final simulation is now looking at the case where the sensors are not placed in a perfect grid in the state space, but are now uniformly randomly placed within the network. The state space setup is identical to the simulation setup posed in the previous section, with hexagonal tessellated cells. Again there are 100 states with 25 sensors in the network. Note that the sensors in this model do not necessarily end up being placed directly on the states of the network, but are merely in the same 2-dimensional continuous space. This arrangement can be seen in Fig. 3.9. Also note that while the sensor positions were picked randomly, once picked the sensors were in the

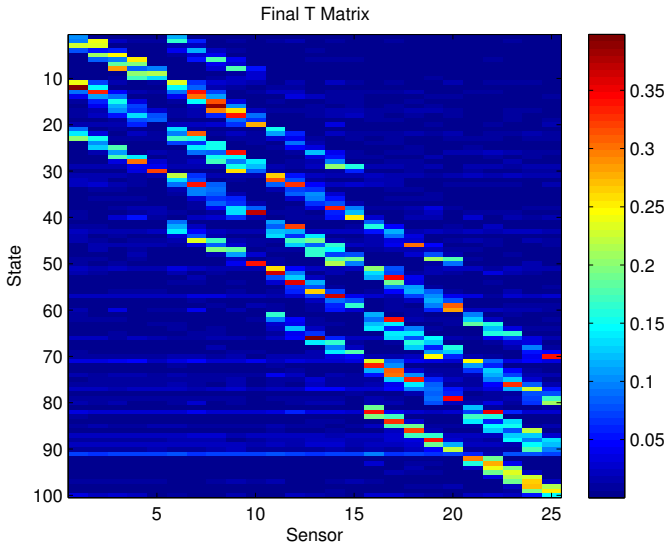


Figure 3.8: Final T^{Δ} Matrix for 2-Dimensional Grid Network with $c = 0.1$

same exact position for all of the simulations.

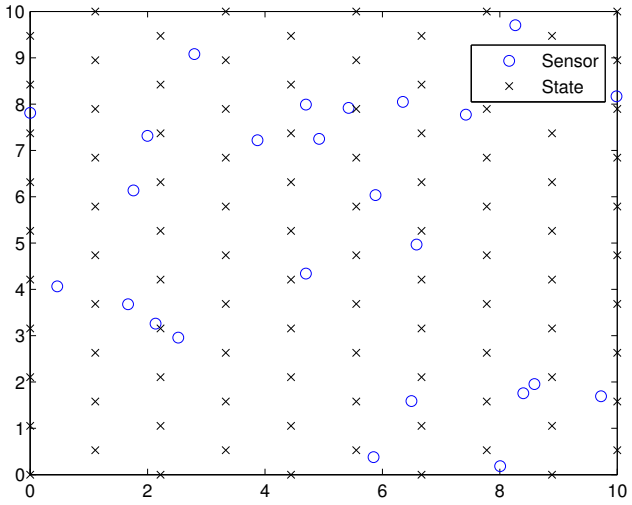


Figure 3.9: 2-Dimensional Random Network Setup

The movement model used in this simulation is completely identical to the one used in the previous simulation, following a random walk behavior with uniform probability movements. The object cannot jump over a cell, moving more than one cell away in any given single step of time.

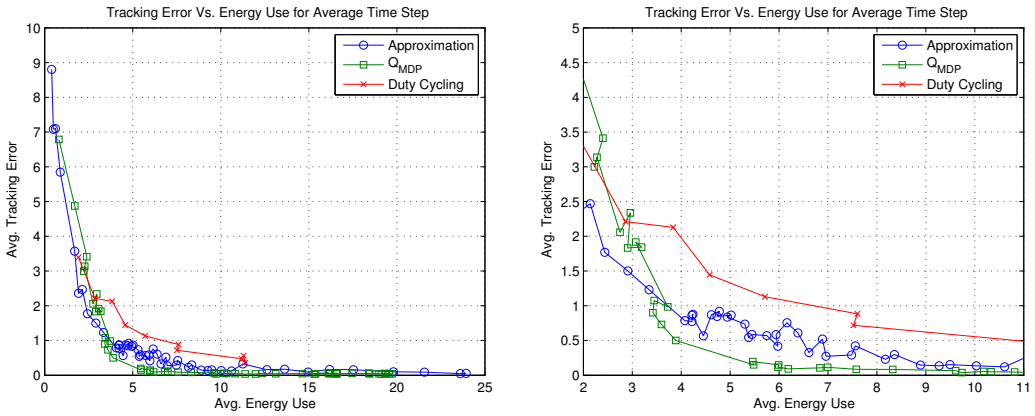
The points on this graph were generated using the same method as in the 1-dimensional array simulation, with each point being the average of 50

simulation runs at that particular cost value c. Again the T^{Δ} matrix is the result of a generating function using 200 Monte Carlo simulations, and then it was trained by running 100 simulations with the learning update code prior to the simulations that actually are used for the data points. Finally, the noise in the system was reduced for this simulation so more separation would appear in the simulation results. For this simulation the additive white Gaussian noise $Z_k \sim \mathcal{N}(0, 0.5)$.

Theoretically this system should behave similarly to the 2-dimensional grid system, with the T^{Δ} matrix reflecting how close given sensors are to given states. The matrix will not have the same order that is present in the grid example though, because the sensor locations and numbers are now completely jumbled (so sensor 1 may be on the opposite end of the network as state 1, whereas sensor 2 could be right on top of it). The random placement of the sensors also would seem to indicate that certain areas in the network will be tracked much more efficiently than certain other areas, because they have relatively higher and lower sensor density per area. This should result in a higher overall tracking error for all the methods tested in the simulation as compared with the previous network layout. The higher tracking error is caused by an inefficiency in the random physical layout of the network rather than an inefficiency associated with any particular control method.

The system behaves much as would be expected in Fig. 3.10. One notable difference that matters more in implementation than in the simulations is the fact that certain sensors appear to be more important overall than other sensors. This is a result of the fact that the sensors are no longer evenly spaced, and the density of sensors per unit area is no longer as constant as it was for the grid network. The end result is that some sensors are closest to a larger area than other sensors, and end up taking more samples on average than other sensors. The more important sensors will likely run out of battery power sooner in implementation than a sensor that is only important to a small area. The implication is that an advantage can be gained by setting up a more evenly distributed network whose sensors are all relatively equal in importance, because this configuration spreads out the power usage better. The data collected shows the same ordering as is seen in the other simulations, although the results are in general messier.

The T^{Δ} matrix for this system is more difficult to analyze than for the other two simulation setups, because now the sensor numbering has no correlation



(a) Zoomed Out (b) Zoomed In

Figure 3.10: 2-Dimensional Random Network Results

to the sensor position in the network, and this means that no convenient pattern emerges in the T^{Δ} matrix mapping as it did previously. The resultant T^{Δ} matrix for $c = 0.1$ can be seen in Fig. 3.11. What can be seen is that the sensors closest to a particular state are still the most important when the object is in that state (although this is not obvious from Fig. 3.11), and so again the intuitive approximation assumptions hold, and the approximation itself works well.

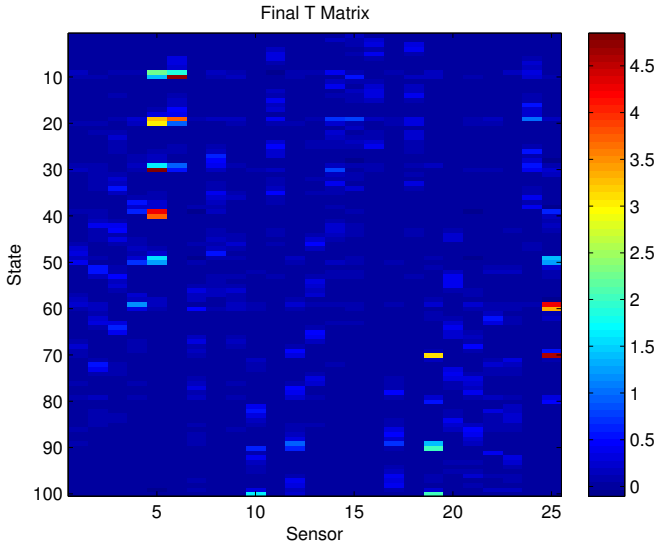


Figure 3.11: Final T^{Δ} Matrix for 2-Dimensional Random Network with $c = 0.1$

It is again seen that the intuitive approximation works very well compared

to the Q_{MDP} solution from [13]. Overall none of the solutions worked as well in the randomized sensor location setup as they did in the grid network setup, which is why the noise power was lowered for the randomized sensor setup. When considering all three setups in turn, it is seen that the intuitive approximation works well in each case, and as such seems to be a good candidate for replacing the Q_{MDP} in WSNs where the computations are very expensive, and the optimal solution is only valued slightly over an approximately optimal solution. This is likely the case in most WSN applications, and so the approximation should be very helpful as an intelligent power control heuristic in implementations.

3.5 Testbed Results

The WSN testbed uses a number of TelosB wireless sensor motes developed by Berkeley [16]. These motes are very low power, and are primarily used for research purposes. The microcontroller used is a 16-bit TI MSP430F1611, and the radio is a 1 mW TI CC2240 which uses the IEEE 802.15.4 standard for communication. Each mote has a sensor suite including an SHT-11 temperature/humidity sensor, and two Hamamatsu photodiodes with different spectral sensitivities. The OS running the hardware is Contiki OS [17], which was developed specifically for applications to low power WSNs. The OS worked very well with the TelosB motes, and provided a simple development environment for the implemented WSN testbed.

The central controller used with the network is a desktop computer running Matlab. The desktop has a TelosB plugged into a USB port to communicate wirelessly with the other motes in the WSN. By creating a simple communication scheme to control the sensors and their sleep times with Matlab, all the difficult work and development could be offloaded to the desktop, allowing for rapid prototyping through the simplified controls provided by the Matlab environment. A real implemented system would work better with a more optimized communication scheme and controller software, but for the purposes of this testbed the Matlab interface proved more than adequate, and very simple to work with.

The role of the mobile object was filled by an iRobot Create, which is basically one of the iRobot 400 vacuum cleaner robots stripped down for use by

researchers and hobbyists. It does not include the vacuum to save on battery life, but does have a tray which was utilized for carrying the actual light source; a cheap desk lamp with a 10 W compact fluorescent bulb. This robot could be controlled via an infrared remote control sold on the iRobot website. The measurements taken by the sensors used the photodiode which was not sensitive to infrared light, to avoid possible interference from the robots remote control. Although not used in these tests, a bluetooth transmitter was also acquired so that movements can be sent from an automated controller, like the desktop running Matlab. This allows for additional versatility in future uses of the robot with a sensor network testbed.

The lab itself was hardly the ideal testing environment for the network, but by placing the WSN on the floor in the middle of the room, and turning off the ambient lights, the network managed to track the object accurately. Tracking can be achieved with the lights on, but it was far more accurate with them off because there is a larger perceived differential by the sensors from the light source. Turning off the ambient lights was lowering the noise floor, and raising the signal-to-noise ratio (SNR) for the testbed observations. SNR is a problem that is worth investigation for specific applications, but the focus of these tests was on simply intelligent sleep control so the problem was handled by simply running the tests in a dark room with the relatively bright light source for tracking. The primary impediment to the development of the testbed was the non-ideal light intensity patterns that resulted from using a real (non-point) light source. For this reason the observation model used by the testbed differed from that used in the simulations, and this is discussed more later on.

3.5.1 1-Dimensional Sensor Array

The 1-dimensional sensor array testbed consisted of 6 sensors laid out in a line with equal spacing, and 11 states setup to be offset from the sensors so the robot does not run into the sensors when moving around the state space. The setup is illustrated in Fig. 3.12, and a picture of the actual testbed in action is seen in Fig. 3.13. The offset nature of the state space from the sensors implies that the tracking should not work as well as if the two were laid on top of each other, as is the case with the 1-dimensional simulation. It

is assumed that the tracking network should still work fairly well; however, and it is worth investigating the algorithm in some non-ideal WSN setups regardless.

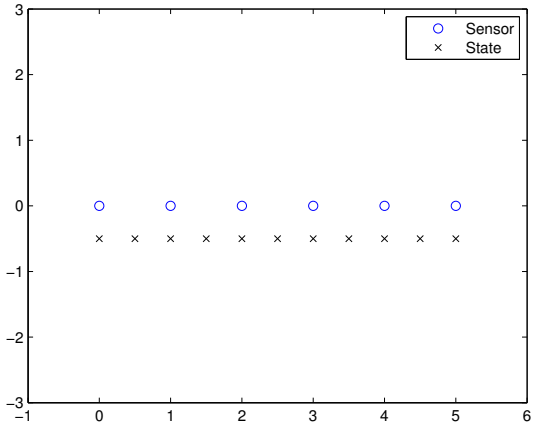


Figure 3.12: 1-Dimensional Network Setup

Figure 3.13: 1-Dimensional WSN Testbed

The movement model used to predict the behavior of the robot was a random walk over the 1-dimensional state space. It is assumed the robot could move to either the state on its immediate left or right (if it exists), or stay in its current state all with uniform probability at each time step. The test lasted for 100 time steps, with a time step duration of 1 second each. The robot started off in state 3 at coordinate (1,-0.5), and moved to the right one state every fifth time step until the robot reached the tenth state. After

this it turned around and went back to state 3 using the same movement pattern. Finally it turned around and moved to state 9.

The observation model used for this testbed implementation was

$$s_{k,l} = \frac{28}{(4d(b_k, l)^2 + 2)} + 2, \tag{3.31}$$

where $d(b_k, l)$ is the distance between the location of the object at time k, b_k, and the sensor l making the observation. The model can be seen compared to the observation model used by the simulations in Fig. 3.14. The observation models for the testbed were changed from the ones used for the simulations because the testbed models better fit actual data seen by the sensors. Note that the light intensity reading observations were made by taking some values at certain distances from the light source, and then subtracting out the readings they reported when the light source was off. The differential is what is used in the testbed setup as a reading for a particular sensor. This eliminates the effects of constant ambient light in the network, and is easier to model.

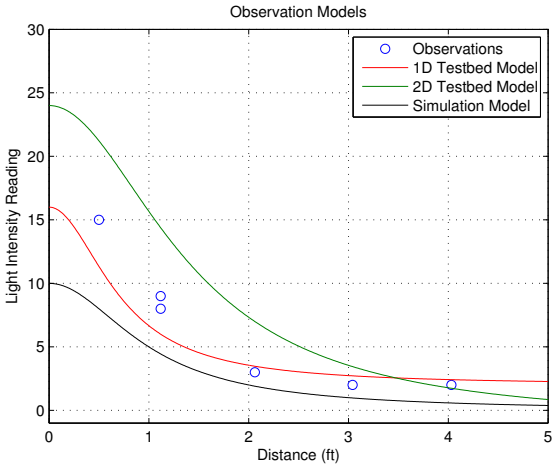


Figure 3.14: Observation Model Comparison

Each point in Fig. 3.15 is the result of a single trial, rather than the average of a number of trials (which was the case for the simulations). This is because of the time-limited nature of running tests on an actual system implementation versus simulations of that system. Figure 3.15 shows that the approximation method detailed in this thesis is outperforming simple duty cycling. It is easy to see when fixing a particular error rate and comparing

the energy usage for the two algorithms at different points. The results show that the network can track the objects movements relatively well, even with as few as one or two sensors active per time step, in both cases. This number should change when considering a more complicated 2-dimensional network setup.

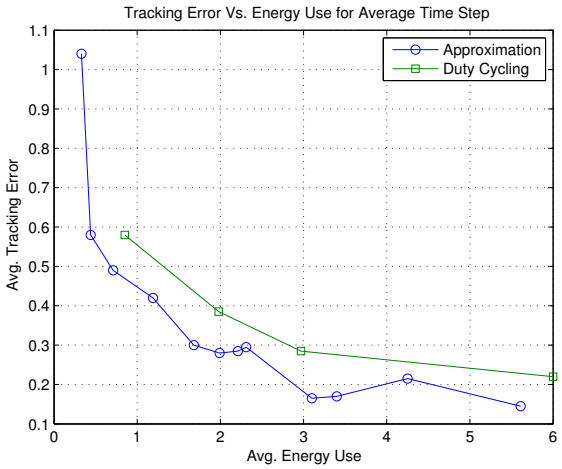


Figure 3.15: 1-Dimensional WSN Testbed Results

The results from this testbed setup show that the approximation method developed does seem to work even in a real system, so the results are positive even if they are not particularly strong. This network setup was very simplistic, and is not a very good representation of what would be expected in a real world scenario, so the next setup looks at a more complicated structure.

3.5.2 2-Dimensional Sensor Grid Array

The testbed 2-dimensional grid network implementation uses a network of 16 sensors arranged in a square grid pattern, which is fewer than the 25 sensors used in the simulation. Again there are four states per sensor in the network, meaning there were 64 states the object could be in at each point in time. The testbed has the sensors arrayed in a square grid, which means the state space is made of slightly distorted hexagons, not quite regular. This will be a source of physical network based error, but it should apply to any power control methods in the same way, and should not be so large that it drastically affects the outcome of the tests. The testbed setup can be seen in Figure 3.16.

Figure 3.16: 2-Dimensional WSN Testbed

The movement model used for the testbed was the same random walk model used in simulations, and the time steps were each one second in duration. This model assumes a higher entropy distribution than the movement of the object actually followed, meaning it errs on the conservative side. In reality the robot could move one cell per second, but could not turn and move that fast, meaning at time k there was a higher probability it would keep going in the same direction or stay in its current cell if it moved. The actual movement pattern followed by the object for the purpose of testing was to stay in a cell for five time units, then move one cell to the North, and repeat until it gets to the edge of the network. At that point the object then turns around and moves South until it hits the other edge, and then repeats this sequence for the duration of the test, which was 100 time units.

The observation model used in this test is given by (3.32) as follows:

$$s_{k,l} = \frac{50}{(d(b_k, l)^2 + 2)} - 1 \tag{3.32}$$

This model also needed to be different from the simulation observation model. A comparison of the models is given by Figure 3.14. It was made to match observations taken from the light source when the 2-dimensional testbed was used, which differed somewhat from the observations seen in Figure 3.14 due to a different light bulb being used.

The points in the Fig. 3.17 are all the result of a single trial, not an average of trials as is the case in the simulations. This figure illustrates that the

intuitive approximation power control method worked better than the duty cycling method over the entire domain of average energy use. The results are not as strong as would be desired, but it is presumable that part of this is due to the fact that the network was so small. It seems from the testbed and simulation results that for effective tracking in a 2-dimensional WSN at least 4-5 sensors are needed to be active each time step for the intelligent power control methods. The results look stronger in the simulations because the network is larger, and duty cycling ends up being even less intelligent. In a smaller network, like the testbed, all of the sensors are somewhat important, and so duty cycling does not lose as much. It is expected that this effect would be magnified further for an even larger state space, and the intuitive approximation method would perform even better relative to duty cycling than is pictured in Fig. 3.17.

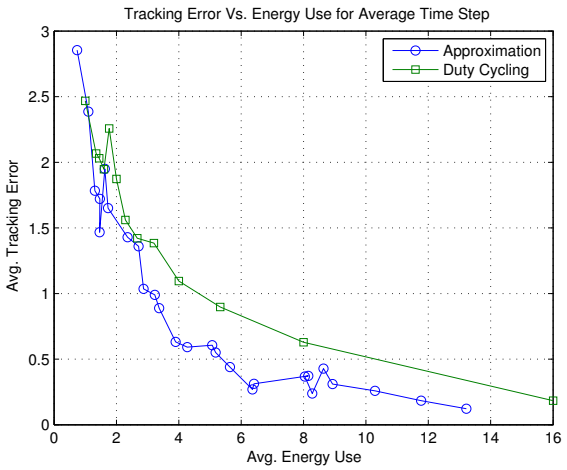


Figure 3.17: 2-Dimensional WSN Testbed Results

From the results of the testbed it is easy to see that the intuitive approximation method of power control for the sensors definitely improves upon simple duty cycling in a WSN. The results were not as strongly demonstrated as would be desired, but it is assumed that the gains would scale up well with a larger network, and be more noteworthy. Considering how easy it was to implement the intuitive approximation method for power control, there are no particularly strong reasons to favor duty cycling in real world implementations of WSNs where a central controller is readily available. This makes the intuitive control useful in the immediate present as a design heuristic, and possibly in the future as well. Further extensions of the

design could likely exploit the setups for specific applications of WSNs even better, further improving results. A few ideas for these kinds of extensions will be briefly explored in the concluding chapter.

3.6 Conclusion

The approximation developed here for the energy efficient tracking problem is a simple idea that performed well in all the simulations and the testbed. It is a control design heuristic that is immediately applicable in a variety of situations, and should be quite easy to extend to more complicated networks than are explored in this work. This is an idea that is well worth additional research and investigation.

At the very least the implementation should be expanded to the case of tracking multiple objects in a sensor network, as is explored mathematically and in simulation in [12], and in much larger sensor networks where the benefits of intelligent tracking can be better seen. If the testbed were expanded to a 10×10 or a 15×15 grid of sensors then the performance should be improved enough over simple duty cycling to well demonstrate to the public how useful and powerful the concept of intelligent controlled sensing for energy efficiency can be.

CHAPTER 4

DIFFUSING POINT SOURCE LOCALIZATION AND INTENSITY TRACKING

In this chapter the goal is to use a sensor network to localize a fixed diffusing point source emitter, and track its output intensity over time. The physical model of a diffusing point source can be applied to a number of different scenarios, but the largest immediate application is likely air quality monitoring, where the location and time varying intensity of the polluting source are of interest.

The chapter begins with a brief problem setup and algorithm description as given in [14]. These sections are followed by a section discussing simulated results, and a section presenting the implemented testbed for this problem. The testbed uses heat instead of carbon dioxide due to the limitations of the hardware available, and health concerns.

4.1 Problem Setup

The basic diffusion equation can be written as follows:

$$\frac{\partial C(x,y,t)}{\partial t} = \alpha \nabla^2 C(x,y,t) + f(x,y,t) \tag{4.1}$$

where $C(x,y,t)$ is the concentration of our emission as a function of the location (x,y) and time t, α is the diffusion coefficient, ∇^2 denotes the Laplacian operator, and $f(x,y,t)$ is the source term. The sensing medium Ω is assumed to be a bounded 2-dimensional space, with boundary $\partial\Omega$ assumed to be a constant. Ω is shaped as a square, with the sensors placed in a square grid at the center. Additionally Ω is assumed to be the same constant at $t = 0$ everywhere. This constant can be assumed to be, without loss of generality, equal to 0. The medium noise is assumed to be mean zero white i.i.d. Gaussian noise with variance σ_a^2.

The network consists of n_0 sensors using $\boldsymbol{x}_i$ to denote the location of the i^{th} sensor. The measurements taken by the sensors have an mean zero additive i.i.d. Gaussian noise term with variance σ_m^2. The vector of all the measurements taken by the sensors at time k is denoted as $\boldsymbol{Y}_k$.

It is assumed the source begins emitting at time $t = 0$, and the unknown fixed source location is denoted as $\boldsymbol{\theta} = (\theta_1, \theta_2)$. During each sampling interval it is assumed the source intensity is constant. The intensity process $\{I_k\}$ forms a Gauss-Markov process so that $I_{k+1} = I_k + W_s(k)$ where W_s are i.i.d. zero mean Gaussian random variables with variance σ_s^2.

Using Galerkin's approximation (as discussed in [14]) a state space representation of the system (4.1) can be developed. Using l_0 deterministic orthonormal functions $\phi_l(\boldsymbol{x})$, $l = 1, ..., l_0$, which are 0 along $\partial\Omega$ as basis functions (4.1) can be described as

$$\tilde{C}(\boldsymbol{x}, t) = \sum_{l=1}^{l_0} Z_{l,t}\phi_l(\boldsymbol{x}), \tag{4.2}$$

where $Z_{l,t}$ is a random process indexed by t, and is such that for $l = 1, ..., l_0$

$$\left\langle \phi_l(t), -\frac{\partial \tilde{C}(\boldsymbol{x}, t)}{\partial t} + \alpha\nabla^2\tilde{C}(\boldsymbol{x}, t) + N_a(\boldsymbol{x}, t) + \sum_{i=1}^{\infty} I_i p(t - i + 1)\delta(\boldsymbol{x} - \boldsymbol{\theta}) \right\rangle = 0, \tag{4.3}$$

where the inner product is defined as $\langle C'(\boldsymbol{x}), C(\boldsymbol{x})\rangle = \int_\Omega C'(\boldsymbol{x})C(\boldsymbol{x})d\boldsymbol{x}$. Substituting $\tilde{C}(\boldsymbol{x}, t)$ from (4.2) gives

$$\frac{\partial Z_{l,t}}{\partial t} = \sum_{m=1}^{l_0} \alpha d(l, m)Z_{m,t} + \sum_{i=1}^{\infty} I_i\phi_l(\boldsymbol{\theta})p(t - i + 1) + N_l(t),$$

with $Z_{l,0} = 0$, and the notation

$$\begin{aligned} N_l(t) &= \langle \phi_l(\boldsymbol{x}), N_a(\boldsymbol{x}, t)\rangle \\ d(l, m) &= \left\langle \phi_l(\boldsymbol{x}), \nabla^2\phi_m(\boldsymbol{x})\right\rangle. \end{aligned}$$

$N_a(\boldsymbol{x}, t)$ is white, meaning $N_l(t)$ will be white as well with power spectral density σ_a^2. Additionally $N_m(t)$ and $N_l(t)$ are independent when $m \neq l$. So

the following system of stochastic differential equations results:

$$\frac{\partial Z_t}{\partial t} = \boldsymbol{D}\boldsymbol{Z}_t + \sum_{i=1}^{\infty} \boldsymbol{\Lambda}(\boldsymbol{\theta}) I_i p(t-i+1) + \boldsymbol{N}(t), \tag{4.4}$$

with $\boldsymbol{Z}_0 = 0$, when

$$\begin{aligned} \boldsymbol{D}(l,m) &= \alpha d(l,m) \\ \boldsymbol{Z}_t &= [Z_{1,t} ... Z_{l_0,t}]^T \\ \boldsymbol{\Lambda}(\boldsymbol{\theta}) &= [\phi_1(\boldsymbol{\theta}) ... \phi_{l_0}(\boldsymbol{\theta})]^T \\ \boldsymbol{N}(t) &= [N_1(t) ... N_{l_0}(t)]^T . \end{aligned}$$

Solving (4.1) gives

$$\boldsymbol{Z}_t = e^{t\boldsymbol{D}} \left(\int_0^t e^{-u\boldsymbol{D}} \left(\boldsymbol{N}(u) + \sum_{i=1}^{\infty} \boldsymbol{\Lambda}(\boldsymbol{\theta}) I_i p(u-i+1) \right) du \right). \tag{4.5}$$

Defining $\boldsymbol{\Phi}$ to be the matrix with elements $\boldsymbol{\Phi}(i,j) = \phi_j(\boldsymbol{x}_i)$, (4.5) can be written recursively as follows:

$$\boldsymbol{Z}_{k+1} = e^{\boldsymbol{D}} \boldsymbol{Z}_k + \left(e^{\boldsymbol{D}} - \boldsymbol{I}_d \right) \boldsymbol{D}^{-1} \boldsymbol{\Lambda}(\boldsymbol{\theta}) I_{k+1} + \boldsymbol{W}_a(k) \tag{4.6}$$

$$I_{k+1} = I_k + W_s(k) \tag{4.7}$$

$$\boldsymbol{Y}_k = \boldsymbol{\Phi}\boldsymbol{Z}_k + \boldsymbol{W}_m(k), \tag{4.8}$$

with $\boldsymbol{W}_m(k)$, $\boldsymbol{W}_a(k)$, and $W_s(k)$ all being mutually independent and i.i.d. Gaussian random processes with covariances $\sigma_m^2 \boldsymbol{I}_d$, $\sigma_a^2 \left(e^{\boldsymbol{D}+\boldsymbol{D}^T} - \boldsymbol{I}_d \right) (\boldsymbol{D} + \boldsymbol{D}^T)^{-1}$, and σ_s^2 respectively. $\boldsymbol{I}_d$ is defined as the identity matrix of the appropriate size.

The approximate state space model for the system given by (4.6), (4.7), and (4.8) only has unknowns $\boldsymbol{\theta}$, and $\{I_k\}$. Developing this model and an algorithm to find these unknowns is the work done in [14].

4.2 Algorithm Description

In [14] the author used the recursive prediction error (RPE) algorithm, given in [15], to estimate the unknown $\boldsymbol{\theta}$ at each time step. When the source

location is known, the solution to the problem of tracking $\{I_k\}$ is to apply a Kalman filter using the following state update equation:

$$\begin{aligned}
\begin{bmatrix} \boldsymbol{Z}_{k+1} \\ I_{k+1} \end{bmatrix} &= \begin{bmatrix} e^{\boldsymbol{D}} & \left(e^{\boldsymbol{D}} - \boldsymbol{I}_d\right) \boldsymbol{D}^{-1} \boldsymbol{\Lambda}(\boldsymbol{\theta}) \\ 0 & 1 \end{bmatrix} \begin{bmatrix} \boldsymbol{Z}_k \\ I_k \end{bmatrix} + \begin{bmatrix} \tilde{\boldsymbol{W}}_a(k) \\ W_s(k) \end{bmatrix} \\
\boldsymbol{Y}_k &= \begin{bmatrix} \boldsymbol{\Lambda}^T & 0 \end{bmatrix} \begin{bmatrix} \boldsymbol{Z}_k \\ I_k \end{bmatrix} + \boldsymbol{W}_m(k), \qquad (4.9)
\end{aligned}$$

where $\tilde{\boldsymbol{W}}_a(k) = \left(e^{\boldsymbol{D}} - \boldsymbol{I}_d\right) \boldsymbol{D}^{-1} \boldsymbol{\Lambda}(\boldsymbol{\theta}) W_s(k) + \boldsymbol{W}_a(k)$.

From (4.8) the one step predictor for the measurement is given by

$$\begin{aligned}
\hat{\boldsymbol{Z}}_{k+1} &= e^{\boldsymbol{D}} \hat{\boldsymbol{Z}}_k + \left(e^{\boldsymbol{D}} - \boldsymbol{I}_d\right) \boldsymbol{D}^{-1} \boldsymbol{\Lambda}(\boldsymbol{\theta}) I_{k+1} + B \left(\boldsymbol{Y}_k - \hat{\boldsymbol{Y}}_k\right) \\
\hat{\boldsymbol{Y}}_{k+1} &= \boldsymbol{\Phi} \hat{\boldsymbol{Z}}_{k+1}, \qquad (4.10)
\end{aligned}$$

where B is the steady state Kalman gain for the system with $\boldsymbol{Z}_k$ as the state and I_k as the input. Applying the RPE algorithm to (4.10) yields

$$\begin{aligned}
\hat{\boldsymbol{Z}}_{k+1} &= e^{\boldsymbol{D}} \hat{\boldsymbol{Z}}_k + \left(e^{\boldsymbol{D}} - \boldsymbol{I}_d\right) \boldsymbol{D}^{-1} \boldsymbol{\Lambda}(\hat{\boldsymbol{\theta}}_k) I_{k+1} + B \boldsymbol{\epsilon}_k && (4.11) \\
\hat{\boldsymbol{Y}}_{k+1} &= \boldsymbol{\Phi} \hat{\boldsymbol{Z}}_{k+1} && (4.12) \\
\boldsymbol{\epsilon}_{k+1} &= \boldsymbol{Y}_{k+1} - \hat{\boldsymbol{Y}}_{k+1} && (4.13) \\
M_k &= \left(e^D - \boldsymbol{I}_d\right) D^{-1} \left(\frac{\partial \boldsymbol{\Lambda}}{\partial \boldsymbol{\theta}}\right)_{\boldsymbol{\theta} = \hat{\boldsymbol{\theta}}_k} I_{k+1} && (4.14) \\
W_{k+1} &= \left(e^D - B\boldsymbol{\Phi}\right) W_k + M_k && (4.15) \\
\Psi_{k+1} &= W_{k+1}^T \boldsymbol{\Phi}^T && (4.16) \\
L_{k+1} &= L_k + \gamma_0(k+1) \left[\boldsymbol{\epsilon}_{k+1} \boldsymbol{\epsilon}_{k+1}^T - L_k\right] && (4.17) \\
R_{k+1} &= R_k + \gamma_1(k+1) \left[\Psi_{k+1} L_{k+1}^{-1} \Psi_{k+1}^T - R_k\right] && (4.18) \\
\hat{\boldsymbol{\theta}}_{k+1} &= \hat{\boldsymbol{\theta}}_k + \gamma_2(k+1) R_{k+1}^{-1} \Psi_{k+1} L_{k+1}^{-1} \boldsymbol{\epsilon}_{k+1}, && (4.19)
\end{aligned}$$

where the parameters are as given in [14].

The algorithm used in [14] is simply to use the current estimate for the source location to predict the next intensity value. Then use the prediction of the next intensity value and the new measurement to update the source location and intensity estimates.

4.3 Simulation Results

Simulations of the system used a sensor network layout of $n_0 = 25$ sensors arranged in a square grid pattern in the middle of the medium space Ω, the same layout used for the simulations in [14]. A visualization of this layout is given in Fig. 4.1.

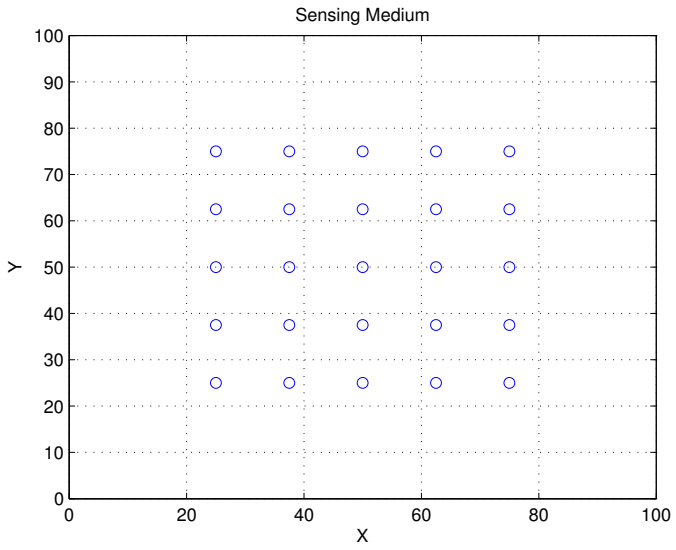


Figure 4.1: Simulation Sensor Network Layout

As in the simulation done for [14], the model parameters are set as $\alpha = 1$, $\sigma_m^2 = 0$, $\sigma_a^2 = 1$, $\sigma_s^2 = 1$, and the state is initialized as $I_0 = 100$, $\boldsymbol{Z}_t = [0, ..., 0]^T$. The 100 orthogonal basis functions used are:

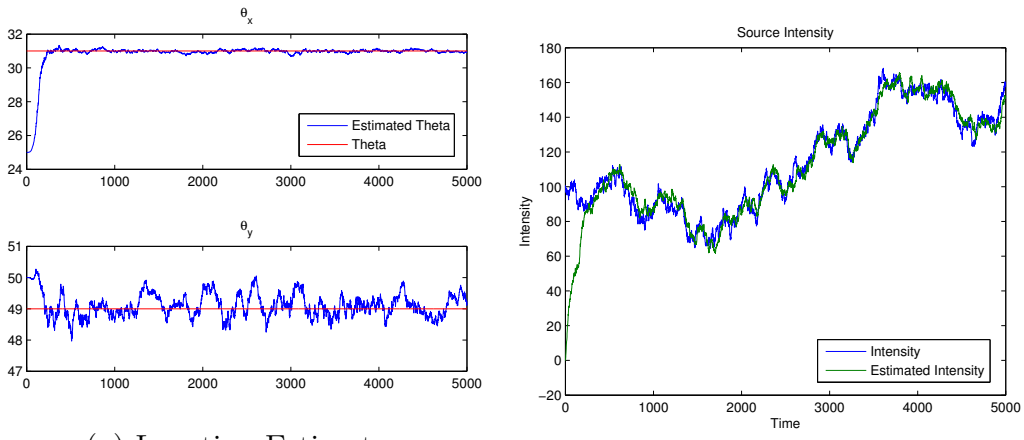
$$\phi_{m,n}(x, y) = \frac{1}{100} \sin \frac{m\pi x}{100} \sin \frac{n\pi y}{100}, m = 1, ..., 10, n = 1, ..., 10. \tag{4.20}$$

The gain sequences used by the RPE algorithm were $\gamma_1(k) = \gamma_0(k) = \frac{1}{k+1}$ and $\gamma_2 = \frac{1}{100}$.

4.3.1 Numerical Results

The results of one run of the simulation are presented below. The estimate for the source location and true source location over time are shown in Fig. 4.2a. Fig. 4.2b shows the estimated and true intensities over time.

From these plots of the results it is obvious that this instance of the algorithm seems to work about as well as it did in [14]. The estimated $\boldsymbol{\theta}$ values



(a) Location Estimate

(b) Intensity Estimate

Figure 4.2: Simulation Results

converge to the true source location within roughly 100-200 samples, and the $\hat{I}(k)$ values converge to the true $I(k)$ values in a similar time span. This basic setup for the algorithm is then the starting point for the code that is used in the implementation of the WSN.

4.4 Testbed Results

The testbed consisted of $n_0 = 16$ sensors arranged into a square grid pattern, very similar to the simulation setup (except with fewer sensors). This arrangement is visualized in Fig. 4.3. The sensors used are the same TelosB motes [16] that have been used for the other chapters of this thesis, running the same Contiki operating system [17], albeit with different application code. Additionally, instead of light, heat coming from the lamp was the sensing medium used. The SHT11 temperature sensor used on the motes proved to be quite sensitive to ambient changes in the air, and could detect the lamps heat after it was turned on for a while. A picture of the testbed can be seen in Fig. 4.4.

In the implementation of the system the arbitrary units used in the simulations must be correlated to reality, and this involved a couple different steps. The first thing that was needed to fit a physical implementation was that the diffusion coefficient α needed to represent thermal diffusivity in air, which is 19 $\frac{mm^2}{s}$. The implementation code was heavily based on the arbitrary units

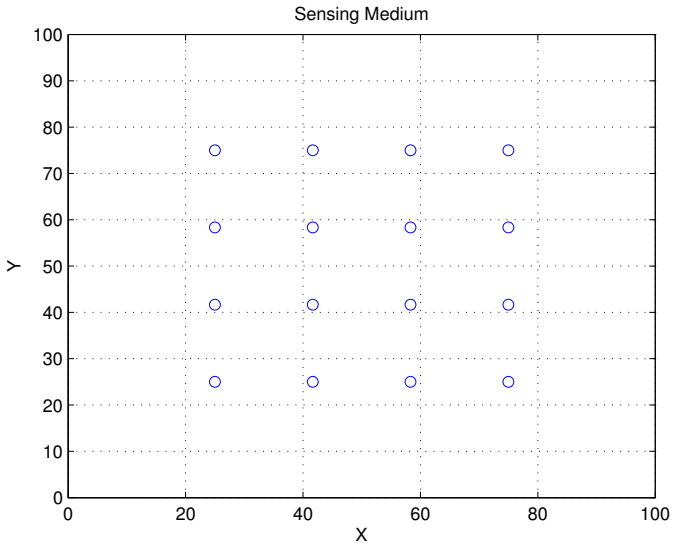


Figure 4.3: Testbed Sensor Network Layout

used in the simulation code, so it was deemed easier to convert the actual α value into the arbitrary units used in the code, rather than converting all the code to use units analogous to the physical world. The spacing between the sensors in the code is $\frac{50}{3}$ units, and the actual spacing for the testbed is 4 in. or 101.6 mm. This means our arbitrary unit thermal coefficient, α_a, is given by

$$\begin{aligned} \frac{101.6mm}{\frac{50}{3}units} &= \frac{\sqrt{19(\frac{mm^2}{s})}}{\sqrt{\alpha_a(\frac{units^2}{s})}} \\ \alpha_a &= \left(\frac{50\sqrt{19}}{3 \times 101.6}\right)^2 \frac{units^2}{s} = 0.5113\frac{units^2}{s}, \end{aligned} \tag{4.21}$$

and then (4.21) is actually adjusted to account for the fact that samples are taken roughly every 5 seconds on average, rather than every second, so that $\alpha_a = 2.5565\frac{units^2}{5s}$.

With this change the code has accounted for the physical model, but it still does not account for the detail that the model actually assumes a zero boundary condition for the medium, Ω. The sensors will report back a number based off of the temperature, and it is NOT normalized on board such that the room temperature is equivalent to an intensity of 0. This is handled in the code at every time step by using the minimum reported temperature

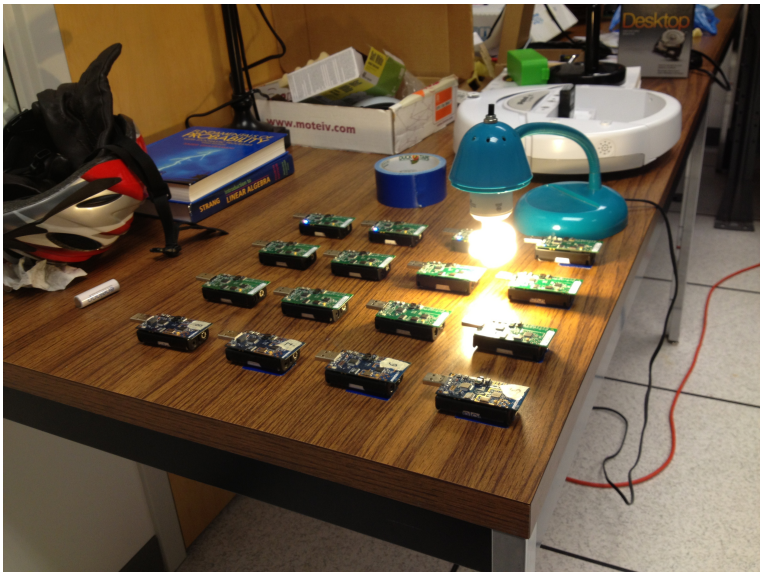


Figure 4.4: Implementation Testbed

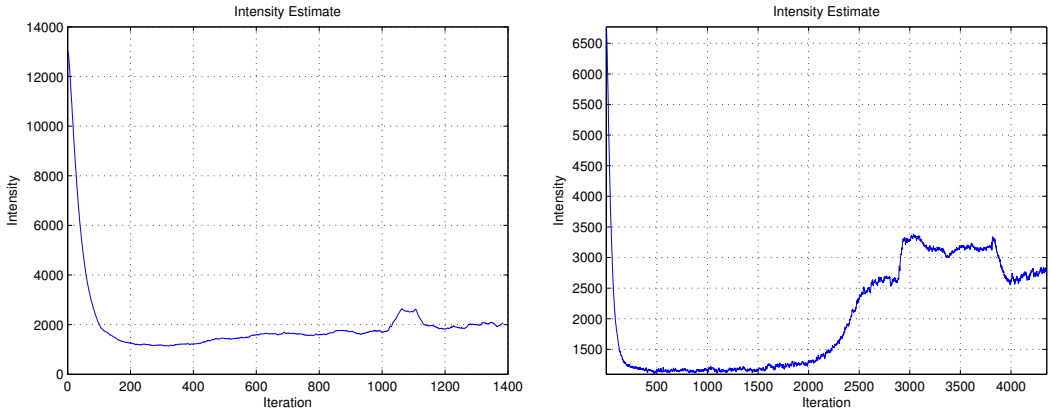
from the sensors as the ambient level. The ambient level is then subtracted from each sensors reading so that the values now report the differential between that sensor and the minimum level for that time step. This implies that one reading is always 0 (the sensor with the minimum heat intensity observation at time k). This operation then approximates the zero boundary level condition, and has the added benefit of making the network somewhat robust to changes in ambient temperature in the air over the time span of the test. It does imply, however, that the estimated intensity $\hat{I}$ given by the network is actually a differential from the ambient reading at each time step k. The total intensity could then be recovered if the ambient level is recorded in memory at every time step k.

One final noted difference is that in (4.20) $\gamma_2 = \frac{1}{100}$, but in the testbed this caused a rather slow convergence for the $\hat{\boldsymbol{\theta}}$ values, so the parameter was increased to $\gamma_2 = \frac{1}{10}$ instead for Test A. The results of this change are what would be expected; the convergence to the true $\boldsymbol{\theta}$ was quicker, but less stable.

4.4.1 Numerical Results

The numerical results from the testbed are less ideal than those displayed by the simulated system, as would be expected. The initial estimate for $\hat{I}$ seems to be quite inaccurate, although the convergence was quick enough

that $\hat{I}$ reduced to a reasonable value within just about 100 iterations of the algorithm. Obviously if a better estimator were obtained the convergence time would likely be shortened by a fair amount. Additionally it would have been desirable for the test to have lasted longer, but what is shown seems to give results consistent with the expected behavior of the system.



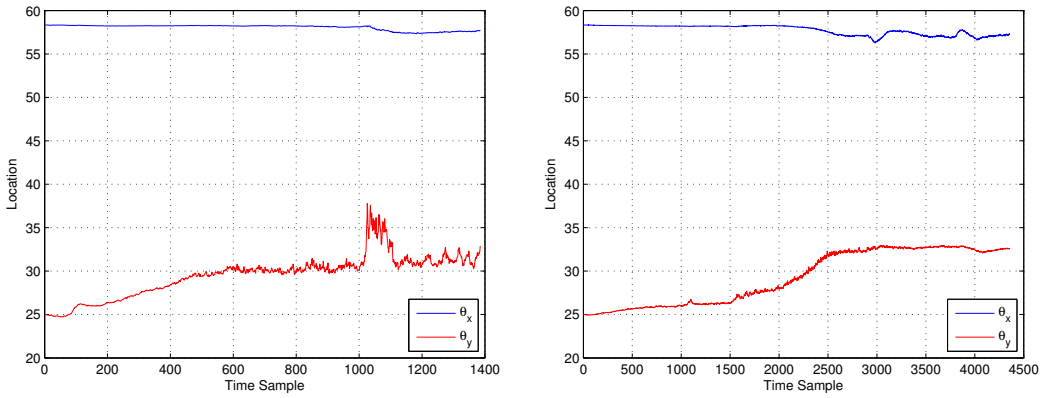
(a) Test A (b) Test B

Figure 4.5: Implementation Intensity Tracking

The behaviors of Fig. 4.5a and Fig. 4.6a are explained as follows. The lamp was turned on right at the beginning of the test, and took a while to heat up to its equilibrium value. At around $k = 1000$ samples a door to the lab room was opened, and both the $\hat{I}$ values and the $\hat{\theta}_y$ values were thrown off for about 80 samples before the algorithm corrected them again. The true position for the lamp in the network was $(x, y) = (58, 33)$, and the values for $\hat{\boldsymbol{\theta}}$ seem to converge to this over the duration of the test. The intensity values are measured in $\frac{1}{1000}$ °C [22], so if the average value for $\hat{I}$ hovers around 2000 it implies the emitting source is about 2 °C above the ambient temperature of the room (approximated by the lowest temperature reading from the network at each time k).

Test B (depicted in Fig. 4.5b and Fig. 4.6b) was similar to Test A, but $\gamma_2 = \frac{1}{100}$ as in the simulations, unlike in Test A where $\gamma_2 = \frac{1}{10}$ for the RPE algorithm. The initial estimate for $\hat{I}$ was again way off, but quickly reduced to close to zero. This is obviously wrong as well, given the last test it would be expected for I to be around 2000 again. This discrepancy can be explained by looking at the $\hat{\boldsymbol{\theta}}$ values for this time frame. The estimate for θ_y did not seem to converge correctly until around $k = 2500$, after which $\hat{I}$ moved to

the expected value. It is likely that the lower value for γ_2 had something to do with this slowdown in convergence, but it is not clear that this is the case. Ultimately Test B did correctly converge, but it did not do so in the expected way, and this is an undesirable result.



(a) Test A (b) Test B

Figure 4.6: Implementation $\hat{\boldsymbol{\theta}}$ Values

Finally depicted in Fig. 4.7 is the estimated value for the medium, $\hat{C}$, as given in (4.2) at the end of Test A. This clearly shows how the thermal intensity of the medium varied over the network space, and the peak is centered right around the true source location. Although the visualization of $\hat{C}$ is quite useful for evaluation of the testbed, it would not necessarily be useful as a replacement for the RPE algorithm to find $\hat{\boldsymbol{\theta}}$. This stems from the fact that computing $\hat{C}$ for all of the space at each time step could be infeasible for some implementations of the algorithm with less computational power in the central controller than the testbed had available.

4.5 Conclusion

The algorithm as presented in [14] is a powerful one, and an interesting one, but the implementation did not work quite as well as was expected. The code could very likely be improved upon, and this may improve the performance of the implementation as well, but it is hard to say for sure. It is not known why the initial estimate for the source intensity is so far off in the implementation, and the uneven convergence of the $\hat{\boldsymbol{\theta}}$ value in Test B is not well understood

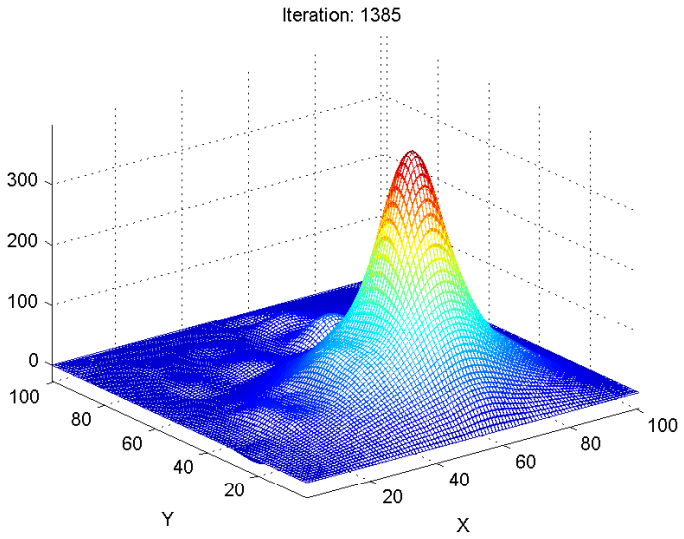


Figure 4.7: Implementation $\tilde{C}(\boldsymbol{x}, t)$

either. Much work remains getting this code up to par.

Once the code is functioning in the implementation as well as it is in the simulation, there still remains some interesting work on the problem. The advection term used in [14] was not used at all in this work. Additionally, tests in a less controlled environment than the lab would be something well worth investigation as well. The belief is that the current mathematical model derived in [14] would be insufficient for application because even if including the advection term the model is not complicated enough to well describe how air pollution would propagate in an urban area. The advection term is assumed to be constant, whereas in reality wind speed and direction are both quite variable over time, and there are physical effects that result from the 3-dimensional space air diffuses in that could not be described in a simple 2-dimensional model. Generalizing the mathematical model is most likely necessary to get this algorithm up to the point where it can be useful in applications.

CHAPTER 5

CONCLUSION

5.1 Summary of Contributions

This thesis examined a number of problems, including a data efficient method of conducting a sequential test, the use of intelligent sleeping methods to extend the battery life of sensors in a WSN for tracking applications, and localizing and tracking the intensity of a fixed point diffusing emitter with an unknown location. The main contributions of this work are listed as follows:

- Developed an implementation of the basic DE-CUSUM algorithm as described in [6].
- Developed a simple approximate solution to the power control problem presented in [13] using intuition derived through the study of the Q_{MDP} solution found in that work.
- Tested the effectiveness of the intuitive approximation compared to the Q_{MDP} solution and simple duty cycling via a number of simulations, with a focus on the 2-dimensional state space.
- Developed a testbed platform to test the approximate solution compared to simple duty cycling.
- Implemented the algorithm given in [14] on the WSN testbed.

5.2 Future Directions

The DE-CUSUM algorithm has a number of extensions that have already been written about for the centralized and decentralized control network case [7], and it would not be difficult to extend the implementation as well

for evaluation. These sorts of change detection networks would give a better idea of when changes are occurring in more complicated systems, like a large bridge, or complicated assembly line equipment. Actual implementation on one of these systems would be feasible, and the performance of this hypothetical network could be a very exciting test case, motivating farther research in the area.

In the energy efficient tracking network there are still issues to be addressed. A full WSN implementation for real world use would likely have many obstructions to the medium of interest (e.g. trees obscuring line of sight, buildings reflecting radio waves) and this would adversely affect the estimated object location. This in turn would lower the quality of the sleep controller's decisions. One advantage of the Q_{MDP} solution over the presented simple approximations is that it has a learning algorithm built into it, so it can account for the obstruction of the medium of interest through the observations made over time. A good idea for a future extension of this work would be to use the fact that the sensors in the network know their relative position, and make the signal strength to distance function dynamic depending on some sufficient statistic of the observations made during the test. In this way the approximation would have its own sort of learning algorithm to compensate for abnormalities in the network.

More complicated models should also be considered in future work that factor in sensors which utilize directional sensing, such as PIR sensors or ultrasonic rangefinders. These directional sensors would violate some core assumptions made in our simple algorithm, and should be handled differently. They would provide more information, and that information should be utilized in an appropriate way.

A lot of work also remains to be done in applying the algorithm to specific applications as well. An example of an interesting test case would be to see if the algorithm could be adapted to fit in a WSN placed in a building. There could be 1-dimensional sub-networks laid out along the hallways, connected to 2-dimensional sub-networks placed in larger areas in the building, such as large rooms or auditoriums. The interesting work would be in stitching together these sub-networks so that accurate estimations could be made when the object moves from one sub-network to another, and how the algorithm should be applied both on the local and global scale.

Fuemmeler and Veeravalli look at the case with possibly multiple objects

in the network being tracked [12], and how best to do this. This is an interesting case, and a simple approximation is not developed for it in this work. It would be interesting to see if the obvious extension of merely doing the approximation for each individual object, then mixing the resulting times at each sensor by some weight, would be effective.

In the diffusing point source localization and intensity tracking chapter, an implementation for the system as given in [14] was developed in a carefully controlled environment, with little ambient disturbance in the medium. The paper allowed for the addition of an advection term to the heat diffusion equation, but this still does not accurately portray the application scenario of tracking a polluting source. Winds are not constant; in fact they change speed and direction quite often, even over a short period of time. This would need to be addressed in the basic mathematical model, and then carried through to the algorithm design and implementation details to get a system that could actually be useful in application. These extensions are worth investigating before the system can find use in its intended application.

5.3 Final Words

In general, engineering research could benefit greatly through more implementation of work on the academic side of the development process. Even a single iteration of the implementation and revision cycle provides feedback to the researchers involved about the real problems with their algorithms, motivating more rigorous and complete development of the solutions derived, and naturally inspiring extensions more directed towards the application details. This extra bit of application focus and demonstrable results would likely encourage more investment in the research from corporations and the government, and more interest from the public as well. This thesis is my small contribution towards that cause.

REFERENCES

[1] J. Polastre, R. Szewczyk, A. Mainwaring, D. Culler, and J. Anderson, "Analysis of wireless sensor networks for habitat monitoring," in *Wireless Sensor Networks*. Springer US, 2004, pp. 399–423.

[2] Y. Li, Z. Wang, and Y. Song, "Wireless sensor network design for wildfire monitoring," in *WCICA 2006*, vol. 1. IEEE, 2006, pp. 109–113.

[3] S. Kim, S. Pakzad, D. Culler, J. Demmel, G. Fenves, S. Glaser, and M. Turon, "Health monitoring of civil infrastructures using wireless sensor networks," in *Information Processing in Sensor Networks, 2007. IPSN 2007. 6th International Symposium on*, April 2007, pp. 254 –263.

[4] K. Chintalapudi, T. Fu, J. Paek, N. Kothari, S. Rangwala, J. Caffrey, R. Govindan, E. Johnson, and S. Masri, "Monitoring civil structures with a wireless sensor network," *Internet Computing, IEEE*, vol. 10, no. 2, pp. 26 – 34, March-April 2006.

[5] T. Banerjee and V. V. Veeravalli, "Data-efficient quickest change detection with on-off observation control," *Sequential Analysis*, vol. 31, no. 1, pp. 40–77, 2012. [Online]. Available: http://www.tandfonline.com/doi/abs/10.1080/07474946.2012.651981

[6] T. Banerjee and V. Veeravalli, "Data-efficient minimax quickest change detection," in *Acoustics, Speech and Signal Processing (ICASSP), 2012 IEEE International Conference on*, March, pp. 3937–3940.

[7] T. Banerjee and V. Veeravalli, "Energy-efficient quickest change detection in sensor networks," in *Statistical Signal Processing Workshop (SSP), 2012 IEEE*, Aug., pp. 636–639.

[8] E. S. Page, "Continuous inspection schemes," *Biometrika*, vol. 41, no. 1/2, pp. pp. 100–115, 1954. [Online]. Available: http://www.jstor.org/stable/2333009

[9] A. Shiryaev, "On optimum methods in quickest detection problems," *Theory of Probability and Its Applications*, vol. 8, no. 1, pp. 22–46, 1963.

[10] V. Veeravalli and J. Fuemmeler, "Efficient tracking in a network of sleepy sensors," in *Proc. IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP'06)*, vol. 5. IEEE, May 2006, pp. 1145–1148.

[11] J. Fuemmeler and V. Veeravalli, "Smart sleeping policies for energy efficient tracking in sensor networks," *IEEE Transactions on Signal Processing*, vol. 56, no. 5, pp. 2091–2101, May 2008.

[12] J. Fuemmeler and V. Veeravalli, "Energy efficient multi-object tracking in sensor networks," *IEEE Transactions on Signal Processing*, vol. 58, no. 7, pp. 3742–3750, July 2010.

[13] J. Fuemmeler, G. Atia, and V. Veeravalli, "Sleep control for tracking in sensor networks," *IEEE Transactions on Signal Processing*, vol. 59, no. 9, pp. 4354–4366, Sept. 2011.

[14] S. Ram and V. Veeravalli, "Localization and intensity tracking of diffusing point sources using sensor networks," in *Global Telecommunications Conference, 2007. GLOBECOM '07. IEEE*, Nov. 2007, pp. 3107–3111.

[15] L. Ljung and T. Söderström, *Theory and Practice of Recursive Identification*. MIT Press, 1983.

[16] J. Polastre, R. Szewczyk, and D. Culler, "Telos: enabling ultra-low power wireless research," in *Information Processing in Sensor Networks, 2005. IPSN 2005. Fourth International Symposium on*, April 2005, pp. 364 – 369.

[17] A. Dunkels, B. Grnvall, and T. Voigt, "Contiki - a lightweight and flexible operating system for tiny networked sensors," in *Proceedings of the First IEEE Workshop on Embedded Networked Sensors (Emnets-I)*, Tampa, Florida, USA, Nov. 2004. [Online]. Available: http://dunkels.com/adam/dunkels04contiki.pdf

[18] D. P. Bertsekas, *Dynamic Programming and Optimal Control*, 3rd ed. Athena Scientific, 2007.

[19] D. Aberdeen, "A (revised) survey of approximate methods for solving partially observable Markov decision processes," National ICT Australia, Canberra, Australia, Tech. Rep., December 2003. [Online]. Available: http://users.rsise.anu.edu.au/ daa/papers.html

[20] M. Littman, A. Cassandra, and L. Kaelbling, "Learning policies for partially observable environments: Scaling up," in *Proc. 12th Int. Conf. Mach. Learn.*, 1995, pp. 362–370.

[21] A. Cassandra, M. Littman, and N. Zhang, "Incremental pruning: A simple, fast, exact algorithm for partially observable markov decision processes," in *Proc. 14th Ann. Conf. Uncert. Artif. Intell.*, 1997, pp. 54–61.

[22] "SHT1x data sheet," Sensirion, Zürich, Switzerland.